The DL_POLY Classic User Manual

W. Smith, T.R. Forester and I.T. Todorov

STFC Daresbury Laboratory
Daresbury, Warrington WA4 4AD
Cheshire, UK

Version 1.7, August 2011

About DL_POLY Classic

DL_POLY Classic is a parallel molecular dynamics simulation package developed at Daresbury Laboratory by W. Smith, T.R. Forester and I.T. Todorov under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5) and the Computational Science and Engineering Department at Daresbury Laboratory. The package is the copyright of the Science Facilities Research Council (STFC) of the United Kingdom. DL_POLY Classic is derived from the DL_POLY_2 package, written by W. Smith and T.R. Forester.

DL_POLY Classic is issued free under a BSD licence, under the terms of which the source code may be freely modified and distributed as long as the copyright statements in the code are retained and proper acknowledgement is made of the authors and Daresbury Laboratory as the place of origin.

The purpose of the DL_POLY Classic package is to provide software for scientific research that is free, accessible and documented.

Disclaimer

None of the authors, nor any of the organisations STFC, EPSRC, CCP5 nor any contributor to the DL_POLY Classic package or its derivatives guarantee that the software and associated documentation is free from error. Neither do they accept responsibility for any loss or damage that results from its use. The responsibility for ensuring that the software is fit for purpose lies entirely with the user.

DL_POLY Classic Acknowledgements

DL_POLY Classic was developed under the auspices of the Science and Technology Facilities Council, the Engineering and Physical Sciences Research Council, and the former Science and Engineering Research Council, under grants from the Computational Science Initiative and the Science and Materials Computing Committee.

Advice, assistance and encouragement in the development of DL_POLY Classic has been given by many people. We gratefully acknowledge the comments, feedback and bug reports from the CCP5 community in the United Kingdom and throughout the world. In addition we thank the following people for contributing code to the package.

1. Maurice Leslie contributed the NOSQUISH rotational algorithm at the heart of many of the rigid body routines.

2. The hyperdynamics algorithms in DL_POLY Classic were developed in a collaboration with Duncan Harris and John Harding at the University of Sheffield and formerly the University of London.

3. The solvation, free energy and solvent induced spectral shift features were developed in collaboration with P.-A. Cazade, P. Bordat and R. Brown at the University of Pau. Travel between Pau and Daresbury by the collaborators was enabled by a grant from the Franco-British Alliance fund.

4. The metadynamics features were developed by David Quigley and Mark Rodger at the University of Warwick.

Manual Notation

In the DL_POLY Classic User Manual specific fonts are used to convey specific meanings:

1. *directories* - itallic font indicate unix file directories
2. ROUTINES - small capitals indicate subroutines, functions and programs.
3. *macros* - sloped text indicates a macro (file of unix commands)
4. **directive** - bold text indicates directives or keywords
5. `variables` - typewriter text indicates named variables and parameters
6. FILE - large capitals indicate filenames.

Contents

The DL_POLY Classic User Manual **a**
About DL_POLY Classic . . . i
Disclaimer . . . ii
Acknowledgements . . . iii
Manual Notation . . . iv

Contents **v**

List of Tables **x**

List of Figures **xi**

1 Introduction **1**
1.1 The DL_POLY Classic Package . . . 3
1.2 Functionality . . . 3
1.2.1 Molecular Systems . . . 3
1.2.2 The DL_POLY Classic Force Field . . . 4
1.2.3 Boundary Conditions . . . 4
1.2.4 The Java Graphical User Interface . . . 4
1.2.5 Algorithms . . . 5
1.3 Programming Style . . . 5
1.3.1 Programming Language . . . 5
1.3.2 Memory Management . . . 6
1.3.3 Target Computers . . . 6
1.3.4 Version Control System (CVS) . . . 6
1.3.5 Required Program Libraries . . . 6
1.3.6 Internal Documentation . . . 6
1.3.7 Subroutine/Function Calling Sequences . . . 7
1.3.8 FORTRAN Parameters . . . 7
1.3.9 Arithmetic Precision . . . 7
1.3.10 Units . . . 7
1.3.11 Error Messages . . . 8
1.4 The DL_POLY Classic Directory Structure . . . 8
1.4.1 The *source* Sub-directory . . . 8
1.4.2 The *utility* Sub-directory . . . 8
1.4.3 The *data* Sub-directory . . . 9
1.4.4 The *bench* Sub-directory . . . 9
1.4.5 The *execute* Sub-directory . . . 9
1.4.6 The *build* Sub-directory . . . 9

1.4.7 The *java* Sub-directory . . . 9
1.5 Obtaining the Source Code . . . 9
1.6 Other Information . . . 10

2 Force Fields and Algorithms 11
2.1 The DL_POLY Classic Force Field . . . 13
2.2 The Intramolecular Potential Functions . . . 15
2.2.1 Bond Potentials . . . 15
2.2.2 Distance Restraints . . . 17
2.2.3 Valence Angle Potentials . . . 17
2.2.4 Angular Restraints . . . 19
2.2.5 Dihedral Angle Potentials . . . 20
2.2.6 Improper Dihedral Angle Potentials . . . 22
2.2.7 Inversion Angle Potentials . . . 23
2.2.8 The Calcite Four-Body Potential . . . 25
2.2.9 Tethering Forces . . . 26
2.2.10 Frozen Atoms . . . 27
2.3 The Intermolecular Potential Functions . . . 28
2.3.1 Short Ranged (van der Waals) Potentials . . . 28
2.3.2 Three Body Potentials . . . 30
2.3.3 The Tersoff Covalent Potential . . . 31
2.3.4 Four Body Potentials . . . 34
2.3.5 Metal Potentials . . . 34
2.3.6 External Fields . . . 41
2.4 Long Ranged Electrostatic (Coulombic) Potentials . . . 42
2.4.1 Atomistic and Charge Group Implementation . . . 43
2.4.2 Direct Coulomb Sum . . . 43
2.4.3 Truncated and Shifted Coulomb Sum . . . 44
2.4.4 Damped Shifted Force Coulomb sum . . . 44
2.4.5 Coulomb Sum with Distance Dependent Dielectric . . . 45
2.4.6 Ewald Sum . . . 46
2.4.7 Smoothed Particle Mesh Ewald . . . 48
2.4.8 Hautman Klein Ewald (HKE) . . . 50
2.4.9 Reaction Field . . . 52
2.4.10 Dynamical Shell Model . . . 53
2.4.11 Relaxed Shell Model . . . 54
2.5 Integration algorithms . . . 54
2.5.1 The Verlet Algorithms . . . 54
2.5.2 Bond Constraints . . . 57
2.5.3 Potential of Mean Force (PMF) Constraints and the Evaluation of Free Energy 59
2.5.4 Thermostats . . . 59
2.5.5 Gaussian Constraints . . . 62
2.5.6 Barostats . . . 63
2.5.7 Rigid Bodies and Rotational Integration Algorithms . . . 67
2.5.8 The DL_POLY Classic Multiple Timestep Algorithm . . . 74
2.6 DL_POLY Parallelisation . . . 74
2.6.1 The Replicated Data Strategy . . . 75
2.6.2 Distributing the Intramolecular Bonded Terms . . . 76
2.6.3 Distributing the Nonbonded Terms . . . 76

2.6.4 Modifications for the Ewald Sum . . . 77
2.6.5 Modifications for SPME . . . 78
2.6.6 Three and Four Body Forces . . . 78
2.6.7 Metal Potentials . . . 78
2.6.8 Summing the Atomic Forces . . . 78
2.6.9 The SHAKE, RATTLE and Parallel QSHAKE Algorithms . . . 79

3 Construction and Execution **81**
3.1 Constructing DL_POLY Classic . . . 83
3.1.1 Overview . . . 83
3.2 Compiling and Running DL_POLY Classic . . . 83
3.2.1 Compiling the Source Code . . . 83
3.2.2 Running DL_POLY Classic . . . 86
3.2.3 Restarting DL_POLY Classic . . . 87
3.2.4 Optimising the Starting Structure . . . 88
3.2.5 Choosing Ewald Sum Variables . . . 89
3.3 DL_POLY Classic Error Processing . . . 92
3.3.1 The DL_POLY Classic Internal Error Facility . . . 92

4 Data Files **93**
4.1 The INPUT files . . . 95
4.1.1 The CONTROL File . . . 95
4.1.2 The CONFIG File . . . 104
4.1.3 The FIELD File . . . 107
4.1.4 The REVOLD File . . . 122
4.1.5 The TABLE File . . . 124
4.1.6 The TABEAM File . . . 125
4.2 The OUTPUT Files . . . 127
4.2.1 The HISTORY File . . . 127
4.2.2 The OUTPUT File . . . 129
4.2.3 The REVCON File . . . 132
4.2.4 The CFGMIN File . . . 132
4.2.5 The REVIVE File . . . 133
4.2.6 The RDFDAT File . . . 133
4.2.7 The ZDNDAT File . . . 133
4.2.8 The STATIS File . . . 134

5 Hyperdynamics **136**
5.1 Overview of Hyperdynamics . . . 138
5.2 The Nudged Elastic Band Calculation . . . 139
5.3 Bias Potential Dynamics . . . 140
5.3.1 Theory of Bias Potential Dynamics . . . 140
5.3.2 Running a BPD Simulation . . . 143
5.3.3 Full Path Kinetics . . . 143
5.3.4 Things to Be Aware of when Running Full Path Kinetics BPD . . . 146
5.3.5 Exploring Configurational Space . . . 147
5.4 Temperature Accelerated Dynamics . . . 147
5.4.1 Theory of Temperature Accelerated Dynamics . . . 147
5.4.2 Running a TAD Simulation . . . 150
5.4.3 Restarting a TAD Simulation . . . 153

5.4.4 Things to Be Aware of when Running TAD . . . 154
5.5 DL_POLY Classic Hyperdynamics Files . . . 154
5.6 Tidying Up the Results of a Hyperdynamics Simulation . . . 157
5.6.1 Refining the Results . . . 157
5.6.2 Treatment of Multiple Maxima in the Reaction Path . . . 158
5.7 Running a Nudged Elastic Band Calculation . . . 158
5.7.1 Things to Aware of when Running a NEB Calculation . . . 159

6 Solvation **161**
6.1 Overview and Background . . . 163
6.2 DL_POLY Energy Decomposition . . . 163
6.2.1 Overview . . . 163
6.2.2 Invoking the DL_POLY Energy Decomposition Option . . . 164
6.2.3 The SOLVAT File . . . 164
6.3 Free Energy by Thermodynamic Integration . . . 166
6.3.1 Thermodynamic Integration . . . 166
6.3.2 Nonlinear Mixing . . . 167
6.3.3 Invoking the DL_POLY Free Energy Option . . . 169
6.3.4 The FREENG File . . . 170
6.4 Solution Spectroscopy . . . 171
6.4.1 Spectroscopy and Classical Simulations . . . 171
6.4.2 Calculating Solvent Induced Spectral Shifts . . . 171
6.4.3 Solvent Relaxation . . . 172
6.4.4 Invoking the Solvent Induced Spectral Shift Option . . . 172
6.4.5 Invoking the Solvent Relaxation Option . . . 173

7 Metadynamics **174**
7.1 Overview . . . 176
7.2 Theory of Metadynamics . . . 176
7.3 Order Parameters . . . 177
7.3.1 Potential Energy as an Order Parameter . . . 177
7.3.2 Steinhardt Order Parameters . . . 178
7.3.3 Tetrahedral Order Parameters . . . 179
7.3.4 Order Parameter Scaling . . . 179
7.4 Running Metadynamics Simulations . . . 179
7.4.1 Additional Considerations . . . 182
7.4.2 Analysing the Metadynamics Results . . . 183

8 The Java Graphical User Interface **185**
8.1 The DL_POLY Java Graphical User Interface . . . 187
8.2 Compiling the Java GUI . . . 187
8.3 Starting the Java GUI . . . 188
8.4 The Graphics Window . . . 189
8.5 The Monitor Window . . . 191
8.6 The GUI Application Menus . . . 191
8.7 File Menu . . . 191
8.7.1 Quit . . . 191
8.7.2 View File . . . 192
8.7.3 Delete File . . . 192
8.7.4 Defaults . . . 192

8.7.5 Reset . . . 193
8.8 FileMaker Menu . . . 193
8.8.1 CONTROL . . . 193
8.8.2 CONFIG . . . 194
8.8.3 FIELD . . . 198
8.8.4 Display . . . 202
8.8.5 Tools . . . 202
8.9 Execute Menu . . . 204
8.9.1 Run DL_POLY . . . 204
8.9.2 Store/Fetch Data . . . 205
8.10 Analysis Menu . . . 206
8.10.1 **Statistics** . . . 206
8.10.2 Structure . . . 207
8.10.3 Dynamics . . . 210
8.10.4 van Hove . . . 211
8.10.5 Display . . . 214
8.10.6 Tools . . . 215
8.11 Information Menu . . . 215
8.12 The GUI Graph Plotter Window . . . 215
8.13 The Molecular Editor . . . 217
8.13.1 Introduction . . . 217
8.13.2 The Buttons of the Molecular Editor . . . 217
8.13.3 The Editor Menu . . . 220
8.13.4 The Sub-Edit Modes . . . 220

9 Example Simulations **224**
9.1 DL_POLY Examples . . . 226
9.1.1 Test Cases . . . 226
9.1.2 Benchmark Cases . . . 231

10 Utilities **233**
10.1 Miscellaneous Utilities . . . 234
10.1.1 Useful Macros . . . 234

Bibliography **240**

Appendices **244**

A The DL_POLY Classic Makefile **244**

B Periodic Boundary Conditions in DL_POLY Classic **247**

C Error Messages and User Action **252**

D Subroutine Locations **322**

Index **331**

List of Tables

4.1 Internal Restart Key . . . 103
4.2 Internal Ensemble Key . . . 103
4.3 Internal Trajectory File Key . . . 103
4.4 Non-bonded force key . . . 103
4.5 CONFIG file key (record 2) . . . 106
4.6 Periodic boundary key (record 2) . . . 106
4.7 Chemical bond potentials . . . 111
4.8 Valence Angle potentials . . . 113
4.9 Dihedral Angle Potentials . . . 114
4.10 Inversion Angle Potentials . . . 115
4.11 Tethering potentials . . . 116
4.12 Definition of pair potential functions and variables . . . 118
4.13 Three-body potentials . . . 119
4.14 Four-body Potentials . . . 119
4.15 Metal Potential . . . 120
4.16 Tersoff Potential . . . 121
4.17 External fields . . . 122

List of Figures

2.1 The interatomic bond vector. . . . 15
2.2 The valence angle and associated vectors . . . 17
2.3 The dihedral angle and associated vectors . . . 20
2.4 The L and D enantiomers and defining vectors . . . 23
2.5 The inversion angle and associated vectors . . . 23
2.6 The vectors of the calcite potential . . . 26
2.7 The SHAKE algorithm . . . 58
2.8 The multiple timestep algorithm . . . 75
2.9 The parallel implementation of the Brode-Ahlrichs algorithm. . . . 77

4.1 DL_POLY Classic input and output files . . . 95

5.1 Model Potential Energy Surface. . . . 138
5.2 Basic NEB Theory . . . 139
5.3 Basic BPD Theory . . . 141
5.4 Basic TAD Theory . . . 149

B.1 The cubic MD cell. . . . 248
B.2 The orthorhomic MD cell. . . . 248
B.3 The parallelepiped MD cell. . . . 249
B.4 The truncated octahedral MD cell. . . . 249
B.5 The rhombic dodecahedral MD cell. . . . 250
B.6 The hexagonal MD cell. . . . 251

Chapter 1

Introduction

Scope of Chapter

This chapter describes the concept, design and directory structure of DL_POLY Classic and how to obtain a copy of the source code.

1.1 The DL_POLY Classic Package

DL_POLY Classic [1] is a molecular simulation package designed to facilitate molecular dynamics simulations of macromolecules, polymers, ionic systems, solutions and other molecular systems on a distributed memory parallel computer. The package was written to support the UK project CCP5 by Bill Smith and Tim Forester [2] under grants from the Engineering and Physical Sciences Research Council and is the copyright of the Science and Technology Facilities Council (STFC).

DL_POLY Classic is based on a replicated data parallelism. It is suitable for simulations of up to 30,000 atoms on up to 100 processors. Though it is designed for distributed memory parallel machines, we have taken care to ensure that it can, with minimum modification, be run on the popular workstations. Scaling up a simulation from a small workstation to a massively parallel machine is therefore a useful feature of the package.

We request that our users respect the copyright of the DL_POLY Classic source and not alter any authorship or copyright notices within.

Further information about the DL_POLY Classic package can be obtained from our website:

http://www.ccp5.ac.uk/DL_POLY_CLASSIC/.

1.2 Functionality

The following is a list of the features DL_POLY Classic.

1.2.1 Molecular Systems

DL_POLY Classic will simulate the following molecular species:

1. Simple atomic systems and mixtures e.g. Ne, Ar, Kr, etc.
2. Simple unpolarisable point ions e.g. NaCl, KCl, etc.
3. Polarisable point ions and molecules e.g. MgO, H_2O etc.D
4. Simple rigid molecules e.g. CCl_4, SF_6, Benzene, etc.
5. Rigid molecular ions with point charges e.g. KNO_3, $(NH_4)_2SO_4$, etc.
6. Polymers with rigid bonds e.g. C_nH_{2n+2}
7. Polymers with rigid bonds and point charges e.g. proteins
8. Macromolecules and biological systems
9. Molecules with flexible bonds
10. Silicate glasses and zeolites
11. Simple metals and alloys e.g. Al, Ni, Cu etc.
12. Covalent systems e.g. C, Si, Ge, SiC, SiGe etc.

1.2.2 The DL_POLY Classic Force Field

The DL_POLY Classic force field includes the following features:

1. All common forms of non-bonded atom-atom potential;
2. Atom-atom (site-site) Coulombic potentials;
3. Valence angle potentials;
4. Dihedral angle potentials;
5. Inversion potentials;
6. Improper dihedral angle potentials;
7. 3-body valence angle and hydrogen bond potentials;
8. 4-body inversion potentials;
9. Finnis-Sinclair and embedded atom type density dependent potentials (for metals) [3, 4].
10. The Tersoff density dependent potential for covalent systems [5].

The parameters describing many of these these potentials may be obtained, for example, from the GROMOS [6], Dreiding [7] or AMBER [8] forcefield, which share functional forms. It is relatively easy to adapt DL_POLY Classic to user specific force fields. Note that DL_POLY Classic does not have its own force field. It is designed to be used with existing force fields.

1.2.3 Boundary Conditions

DL_POLY Classic will accommodate the following boundary conditions:

1. None e.g. isolated polymer in space.
2. Cubic periodic boundaries.
3. Orthorhombic periodic boundaries.
4. Parallelepiped periodic boundaries.
5. Truncated octahedral periodic boundaries.
6. Rhombic dodecahedral periodic boundaries.
7. Slab (x,y periodic, z nonperiodic).
8. Hexagonal prism periodic boundaries.

These are describe in detail in Appendix B.

1.2.4 The Java Graphical User Interface

DL_POLY Classic has a Graphical User Interface (GUI) written specifically for the package in the Java programming language from Sun microsystems. The Java programming environment is free and it is particularly suitable for building graphical user interfaces. An attractive aspect of java is the portability of the compiled GUI, which may be run without recompiling on any Java supported machine. The GUI is an integral component of the DL_POLY Classic package and is available under the same terms. (See [9].)

1.2.5 Algorithms

1.2.5.1 Parallel Algorithms

DL_POLY Classic exclusively employs the **Replicated Data** parallelisation strategy [10, 11] (see section 2.6.1).

1.2.5.2 Molecular Dynamics Algorithms

The DL_POLY Classic MD algorithms are optionally available in the form of the Verlet Leapfrog or the Velocity Verlet integration algorithms [12].

In the leapfrog scheme a parallel version of the SHAKE algorithm [13, 11] is used for bond constraints and a similar adaptation of the RATTLE algorithm [14] is implmented in the velocity Verlet scheme.

Rigid body rotational motion is handled under the leapfrog scheme with Fincham's implicit quaternion algorithm (FIQA) [15]. For velocity Verlet integration of rigid bodies DL_POLY Classic uses the 'NOSQUISH' algorithm of Miller *et al* [16].

Rigid molecular species linked by rigid bonds are handled with an algorithm of our own devising, called the QSHAKE algorithm [17] which has been adapted for both leapfrog and velocity Verlet schemes.

NVE, NVT, NPT and N$\underline{\sigma}$T ensembles are available, with a selection of thermostats and barostats. The velocity Verlet versions are based on the reversible integrators of Martyna *et al* [18].

The NVT algorithms in DL_POLY Classic are those of Evans [19], Berendsen [20]; and Hoover [21]. The NPT algorithms are those of Berendsen [20] and Hoover [21] and the NσT algorithms are those of Berendsen [20] and Hoover [21].

The full range of MD algorithms available in DL_POLY Classic is described in section 2.5.

1.2.5.3 Structure Relaxation Algorithms

DL_POLY Classic has a selection of structure relaxation methods available. These are useful to improve the starting structure of a molecular dynamics simulation. The algorithms available are:

1. 'Zero' temperature molecular dynamics (sometimes called damped molecular dynamics);
2. Conjugate gradients minimisation;
3. 'Programmed' energy minimisation, involving both molecular dynamics and conjugate gradients .

Starting structure minimisation is described in section 3.2.4.

1.3 Programming Style

The programming style of DL_POLY Classic is intended to be as uniform as possible. The following stylistic rules apply throughout. Potential contributors of code are requested to note the stylistic convention.

1.3.1 Programming Language

DL_POLY Classic is written exclusively in FORTRAN 90. Use is made of F90 Modules. Explicit type declaration is used throughout.

1.3.2 Memory Management

In DL_POLY Classic, the major array dimensions are calculated at the start of execution and the associated arrays created through the dynamic array allocation features of FORTRAN 90.

1.3.3 Target Computers

DL_POLY Classic is intended for distributed memory parallel computers. However, versions of the program for serial computers are easily produced. To facilitate this all machine specific calls are located in dedicated FORTRAN routines, to permit substitution by appropriate alternatives.

DL_POLY Classic will run on a wide selection of computers. This includes most single processor workstations for which it requires a FORTRAN 90 compiler and (preferably) a UNIX environment. It has also been compiled for a Windows PC using both the GFORTRAN and G95 FORTRAN compiler augmented by the CygWin UNIX shell. The Message Passing Interface (MPI) software is essential for parallel execution.

1.3.4 Version Control System (CVS)

DL_POLY Classic was developed with the aid of the CVS version control system. We strongly recommend that users of DL_POLY Classic adopt this system for local development of the DL_POLY Classic code, particularly where several users access the same source code. For information on CVS please contact:

$$info - cvs - request@gnu.org$$

or visit the website:

$$http://www.ccp5.ac.uk/DL_POLY_CLASSIC/.$$

1.3.5 Required Program Libraries

DL_POLY Classic is, for the most part, self contained and does not require access to additional program libraries. The exception is the MPI software library required for parallel execution.

Users requiring the Smoothed Particle Mesh Ewald (SPME) method may prefer to use a proprietary 3D FFT other than the one (DLPFFT3) supplied with the package for optimal performance. There are comments in the source code which provide guidance for applications on Cray and IBM computers, which use the routines CCFFT3D and DCFT3 respectively. Similarly users will find comments for the public domain FFT routine FFTWND_FFT.

1.3.6 Internal Documentation

All subroutines are supplied with a header block of FORTRAN COMMENT records giving:

1. The name of the author and/or modifying author
2. The version number or date of production
3. A brief description of the function of the subroutine
4. A copyright statement

Elsewhere FORTRAN COMMENT cards are used liberally.

1.3.7 Subroutine/Function Calling Sequences

The variables in the subroutine arguments are specified in the order:

1. logical and logical arrays
2. character and character arrays
3. integer
4. real and complex
5. integer arrays
6. real and complex arrays

This is admittedly arbitrary, but it really does help with error detection.

1.3.8 FORTRAN Parameters

All global parameters defined by the FORTRAN parameter statements are specified in the module: SETUP_MODULE. All parameters specified in SETUP_MODULE are described by one or more comment cards.

1.3.9 Arithmetic Precision

All real variables and parameters are specified in 64-bit precision (i.e real*8).

1.3.10 Units

Internally all DL_POLY Classic subroutines and functions assume the use of the following defined *molecular units*:

1. The unit of time (t_o) is 1×10^{-12} seconds (i.e. picoseconds).
2. The unit of length (ℓ_o) is 1×10^{-10} metres (i.e. Ångstroms).
3. The unit of mass (m_o) is $1.6605402 \times 10^{-27}$ kilograms (i.e. atomic mass units).
4. The unit of charge (q_o) is $1.60217733 \times 10^{-19}$ coulombs (i.e. unit of proton charge).
5. The unit of energy ($E_o = m_o(\ell_o/t_o)^2$) is $1.6605402 \times 10^{-23}$ Joules (10 J mol^{-1}).
6. The unit of pressure ($\mathcal{P}_o = E_o\ell_o^{-3}$) is 1.6605402×10^{7} Pascal (163.882576 atm).
7. Planck's constant ($\hbar$) which is $6.350780719 \times E_o t_o$.

In addition the following conversion factors are used:
The coulombic conversion factor (γ_o) is:

$$\gamma_o = \frac{1}{E_o}\left[\frac{q_o^2}{4\pi\epsilon_o\ell_o}\right] = 138935.4835$$

such that:

$$U_{MKS} = E_o\gamma_o U_{Internal}$$

Where U represents the configuration energy.
The Boltzmann factor (k_B) is 0.831451115 $E_o K^{-1}$, such that:

$$T = E_{kin}/k_B$$

represents the conversion from kinetic energy (in internal units) to temperature (in Kelvin).

Note: In the DL_POLY Classic CONTROL and OUTPUT files, the pressure is given in units of kilo-atmospheres (k-atm) at all times. The unit of energy is either DL_POLY Classic units specified above, or in other units specified by the user at run time. The default is DL_POLY units.

1.3.11 Error Messages

All errors detected by DL_POLY Classic during run time initiate a call to the subroutine ERROR, which prints an error message in the standard output file and terminates the program. All terminations of the program are global (i.e. every node of the parallel computer will be informed of the termination condition and stop executing.)

In addition to terminal error messages, DL_POLY Classic will sometimes print warning messages. These indicate that the code has detected something that is unusual or inconsistent. The detection is non-fatal, but the user should make sure that the warning does represent a harmless condition.

More on error handling can be found in section (3.3).

1.4 The DL_POLY Classic Directory Structure

The entire DL_POLY Classic package is stored in a Unix directory structure. The topmost directory is named *dl_poly_class*. Beneath this directory are several sub-directories:

sub-directory	contents
source	primary subroutines for the DL_POLY Classic package
utility	subroutines, programs and example data for all utilities
data	example input and output files for DL_POLY Classic
execute	the DL_POLY Classic run-time directory
build	makefiles to assemble and compile DL_POLY Classic programs
java	directory of Java and FORTRAN routines for the Java GUI

A more detailed description of each sub-directory follows.

1.4.1 The *source* Sub-directory

In this sub-directory all the essential source code for DL_POLY Classic, excluding the utility software. The modules are assembled at compile time using an appropriate makefile.

1.4.2 The *utility* Sub-directory

This sub-directory stores all the utility programs in DL_POLY Classic. Users who devise their own utilities are advised to store them in the *utility* sub-directory.

1.4.3 The *data* Sub-directory

This sub-directory contains examples of input and output files for testing the released version of DL_POLY Classic. The examples of input data are copied into the *execute* sub-directory when a program is being tested. The test cases are documented in chapter 9.

1.4.4 The *bench* Sub-directory

This directory contains examples of input and output data for DL_POLY Classic that are suitable for benchmarking DL_POLY Classic on large scale computers. These are described in chapter 9.

1.4.5 The *execute* Sub-directory

In the supplied version of DL_POLY Classic, this sub-directory contains only a few macros for copying and storing data from and to the *data* sub-directory and for submitting programs for execution. (These are decribed in section 10.1.1.) However when a DL_POLY Classic program is assembled using its makefile, it will be placed in this sub-directory and will subsequently be executed from here. The output from the job will also appear here, so users will find it convenient to use this sub-directory if they wish to use DL_POLY Classic as intended. (The experienced user is not absolutely required to use DL_POLY Classic this way however.)

1.4.6 The *build* Sub-directory

This sub-directory contains the standard makefiles for the creation (i.e. compilation and linking) of the DL_POLY Classic simulation programs. The makefiles supplied select the appropriate subroutines from the *source* sub-directory and deposit the executable program in the *execute* directory. The user is advised to copy the appropriate makefile into the *source* directory, in case any modifications are required. The copy in the *build* sub-directory will then serve as a backup.

1.4.7 The *java* Sub-directory

The DL_POLY Classic Java Graphical User Interface (GUI) is based on the Java language developed by Sun. The Java source code for this GUI is to be found in this sub-directory, along with a few FORTRAN sub-sub-directories which contain some additional capabilities accessible from the GUI. These sources are complete and sufficient to create a working GUI, provided the user has installed the Java Development Kit, (1.4 or above) which is available free from Sun at

http://java.sun.com

The GUI, once compiled, may be executed on any machine where Java is installed, though note the FORTRAN components will need to be recompiled if the machine is changed. (See [9].)

1.5 Obtaining the Source Code

DL_POLY Classic source code and the associated test data is available from CCPForge at

http://ccpforge.cse.rl.ac.uk.

1.6 Other Information

The DL_POLY Classic website:

http://www.ccp5.ac.uk/DL_POLY_CLASSIC/

provides additional information in the form of

1. Access to all documentation;
2. Frequently asked questions;
3. Bug reports;
4. Access to the DL_POLY online forum.

The DL_POLY **Forum** is a web based centre for all DL_POLY users to exchange comments and queries. You may access the forum through the DL_POLY Classic website. A registration (and vetting) process is required before you can use the forum, but it is open in principle to everyone. It is a good place to contact other users and discuss applications.

Chapter 2

Force Fields and Algorithms

Scope of Chapter

This chapter describes the interaction potentials and simulation algorithms coded into DL_POLY Classic.

2.1 The DL_POLY Classic Force Field

The force field is the set of functions needed to define the interactions in a molecular system. These may have a wide variety of analytical forms, with some basis in chemical physics, which must be parameterised to give the correct energy and forces. A huge variety of forms is possible and for this reason the DL_POLY Classic force field is designed to be adaptable. While it is not supplied with its own force field parameters, many of the functions familiar to GROMOS, [6] Dreiding [7], AMBER [8] and OPLS [22] users have been coded in the package, as well as less familiar forms. In addition DL_POLY Classic retains the possibility of the user defining additional potentials.

In DL_POLY Classic the total configuration energy of a molecular system may be written as:

$$\begin{aligned} U(\underline{r}_1, \underline{r}_2, \ldots, \underline{r}_N) \;=\; & \sum_{i_{bond}=1}^{N_{bond}} U_{bond}(i_{bond}, \underline{r}_a, \underline{r}_b) \\ & + \sum_{i_{angle}=1}^{N_{angle}} U_{angle}(i_{angle}, \underline{r}_a, \underline{r}_b, \underline{r}_c) \\ & + \sum_{i_{dihed}=1}^{N_{dihed}} U_{dihed}(i_{dihed}, \underline{r}_a, \underline{r}_b, \underline{r}_c, \underline{r}_d) \\ & + \sum_{i_{inv}=1}^{N_{inv}} U_{inv}(i_{inv}, \underline{r}_a, \underline{r}_b, \underline{r}_c, \underline{r}_d) \\ & + \sum_{i=1}^{N-1} \sum_{j>i}^{N} U_{pair}(i, j, |\underline{r}_i - \underline{r}_j|) \\ & + \sum_{i=1}^{N-2} \sum_{j>i}^{N-1} \sum_{k>j}^{N} U_{3_body}(i, j, k, \underline{r}_i, \underline{r}_j, \underline{r}_k) \\ & + \sum_{i=1}^{N-1} \sum_{j>i}^{N} U_{Tersoff}(i, j, \underline{r}_i, \underline{r}_j, \underline{R}^N) \\ & + \sum_{i=1}^{N-3} \sum_{j>i}^{N-2} \sum_{k>j}^{N-1} \sum_{n>k}^{N} U_{4_body}(i, j, k, n, \underline{r}_i, \underline{r}_j, \underline{r}_k, \underline{r}_n) \\ & + \sum_{i=1}^{N} U_{Metal}(i, \underline{r}_i, \underline{R}^N) \\ & + \sum_{i=1}^{N} U_{extn}(i, \underline{r}_i, \underline{v}_i) \end{aligned} \tag{2.1}$$

where U_{bond}, U_{angle}, U_{dihed}, U_{inv}, U_{pair}, U_{3_body}, $U_{Tersoff}$ and U_{4_body} are empirical interaction functions representing chemical bonds, valence angles, dihedral angles, inversion angles, pair-body, three-body, Tersoff (many-body covalent), and four-body forces respectively. The first four are regarded by DL_POLY Classic as *intra*-molecular interactions and the next five as *inter*-molecular interactions. The term U_{metal} is a density dependent (and therefore many-body) metal potential . The final term U_{extn} represents an *external field* potential.

The position vectors $\underline{r}_a, \underline{r}_b, \underline{r}_c$ and $\underline{r}_d$ refer to the positions of the atoms specifically involved in a given interaction. (Almost universally, it is the *differences* in position that determine the interaction.) A special vector $\underline{R}^N$ is used to indicate a many-body dependence. The numbers N_{bond}, N_{angle}, N_{dihed} and N_{inv} refer to the total numbers of these respective interactions present

in the simulated system, and the indices i_{bond}, i_{angle}, i_{inv} and i_{dihed} uniquely specify an individual interaction of each type. It is important to note that there is no global specification of the intramolecular interactions in DL_POLY Classic - all bonds, valence angles and dihedrals must be individually cited.

The indices i, j (and k, n) appearing in the pair-body (and three or four-body) terms indicate the atoms involved in the interaction. There is normally a very large number of these and they are therefore specified according to atom *types* rather than indices. In DL_POLY Classic it is assumed that the pair-body terms arise from van der Waals and/or electrostatic (Coulombic) forces. The former are regarded as short ranged interactions and the latter as long ranged. Long ranged forces require special techniques to evaluate accurately (see section 2.4.) In DL_POLY Classic the three-body terms are restricted to valence angle and H-bond forms. The nonbonded, three-body, four-body and Tersoff , interactions are globally specified according to the *types* of atoms involved. DL_POLY Classic also has the ability to handle metals via density dependent functions (see below). Though essentially many-body potentials their particular form means they are handled in a manner very similar to pair potentials.

In DL_POLY Classic the intramolecular bonded terms are handled using bookkeeping arrays, which specify the atoms involved in a particular interaction and point to the appropriate arrays of parameters that define the potential. The calculation of bonded forces therefore follows the simple scheme:

1. Every atom in the simulated system is assigned a unique index number from 1 to N;
2. Every intramolecular bonded term U_{type} in the system has a unique index number i_{type}: from 1 to N_{type} where *type* represents a bond, angle or dihedral.
3. A pointer array $key_{type}(n_{type}, i_{type})$ carries the indices of the specific atoms involved in the potential term labelled i_{type}. The dimension n_{type} will be 2, 3 or 4, if the term represents a bond, valence angle, dihedral/inversion.
4. The array $key_{type}(n_{type}, i_{type})$ is used to identify the atoms in a bonded term and the appropriate form of interaction and thus to calculate the energy and forces.

DL_POLY Classic calculates the nonbonded pair interactions using a Verlet neighbour list [12] which is reconstructed at intervals during the simulation. This list records the indices of all 'secondary' atoms within a certain radius of each 'primary' atom; the radius being the cut-off radius (r_{cut}) normally applied to the nonbonded potential function, plus an additional increment (Δr_{cut}). The neighbour list removes the need to scan over all atoms in the simulation at every timestep. The larger radius ($r_{cut} + \Delta r_{cut}$) means the same list can be used for several timesteps without requiring an update. The frequency at which the list must be updated depends on the thickness of the region Δr_{cut}. DL_POLY Classic has two methods for constructing the neighbour list: the first is based on the Brode-Ahlrichs scheme [23] and is used when r_{cut} is large in comparison with the simulation cell; the second uses the link-cell algorithm [24] when r_{cut} is relatively small. The potential energy and forces arising from the nonbonded interactions are calculated using interpolation tables.

A complication in the construction of the Verlet neighbour list for macromolecules is the concept of *excluded atoms*, which arises from the need to exclude certain atom pairs from the overall list. Which atom pairs need to be excluded is dependent on the precise nature of the force field model, but as a minimum atom pairs linked via extensible bonds or constraints and atoms (grouped in pairs) linked via valence angles are probable candidates. The assumption behind this requirement is that atoms that are formally bonded in a chemical sense, should not participate in nonbonded interactions with each other. (However this is not a universal requirement of all force fields.) The same considerations are needed in dealing with charged excluded atoms. DL_POLY Classic has

several subroutines available for constructing the Verlet neighbour list, while taking care of the excluded atoms (see chapter 3 for further information.)

Three- and four-body nonbonded forces are assumed to be short ranged and therefore calculated using the link-cell algorithm [24]. They ignore the possibility of there being any excluded interactions involving the atoms concerned.

Throughout this section the description of the force field assumes the simulated system is described as an assembly of atoms. This is for convenience only and readers should understand that DL_POLY Classic does recognise molecular entities, defined either through constraint bonds or rigid bodies. In the case of rigid bodies, the atomic forces are resolved into molecular forces and torques. These matters are discussed in greater detail later in sections 2.5.2.1 and 2.5.7).

2.2 The Intramolecular Potential Functions

In this section we catalogue and describe the forms of potential function available in DL_POLY Classic The **key words** required to select potential forms are given in brackets () before each definition. The derivations of the atomic forces, virial and stress tensor are also outlined.

2.2.1 Bond Potentials

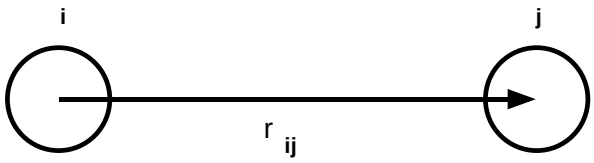

Figure 2.1: The interatomic bond vector.

The bond potentials describe *explicit* bonds between specified atoms. They are functions of the interatomic distance only. The potential functions available are as follows.

1. Harmonic bond: (**harm**)
$$U(r_{ij}) = \frac{1}{2}k(r_{ij} - r_o)^2; \tag{2.2}$$

2. Morse potential: (**mors**)
$$U(r_{ij}) = E_o[\{1 - \exp(-k(r_{ij} - r_o))\}^2 - 1]; \tag{2.3}$$

3. 12-6 potential bond: (**12-6**)
$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{6}}\right); \tag{2.4}$$

4. Restrained harmonic: (**rhrm**)
$$U(r_{ij}) = \frac{1}{2}k(r_{ij} - r_o)^2 \qquad |r_{ij} - r_o| \le r_c; \tag{2.5}$$
$$U(r_{ij}) = \frac{1}{2}kr_c^2 + kr_c(|r_{ij} - r_o| - r_c) \qquad |r_{ij} - r_o| > r_c; \tag{2.6}$$

5. Quartic potential: (**quar**)

$$U(r_{ij}) = \frac{k}{2}(r_{ij} - r_o)^2 + \frac{k'}{3}(r_{ij} - r_o)^3 + \frac{k''}{4}(r_{ij} - r_o)^4. \tag{2.7}$$

6. Buckingham potential: (**buck**)

$$U(r_{ij}) = A\ \exp\left(-\frac{r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^6}; \tag{2.8}$$

7. Shifted finitely extendible non-linear elastic (FENE) potential [25, 26, 27]: (**fene**)

$$U(r_{ij}) = \begin{cases} -0.5\ k\ R_o^2\ ln\left[1 - \left(\frac{r_{ij}-\Delta}{R_o}\right)^2\right] & : \quad r_{ij} < R_o + \Delta \\ \infty & : \quad r_{ij} \geq R_o + \Delta \end{cases} \tag{2.9}$$

   The FENE potential is used to maintain the distance between connected beads and to prevent chains from crossing each other. It is used in combination with the WCA (2.99) potential to create a potential well for the flexible bonds of a molecule, that maintains the topology of the molecule. This implementation allows for a radius shift of up to half a R_o ($|\Delta| \leq 0.5\ R_o$) with a default of zero ($\Delta_{default} = 0$).

8. Coulomb potential: (**coul**)

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{2.10}$$

   Note that the Coulombic bond potential is not normally required, as generally the electrostatic interactions are handled as nonbonded terms elsewhere in the program. However, it is sometimes explicit in the description of the chemical bond in a way that is different from the default electrostatic treatment, and needs to be introduced as an extra feature.

In these formulae r_{ij} is the distance between atoms labelled i and j:

$$r_{ij} = |\underline{r}_j - \underline{r}_i|, \tag{2.11}$$

where $\underline{r}_\ell$ is the position vector of an atom labelled ℓ. [1]

The force on the atom j arising from a bond potential is obtained using the general formula:

$$\underline{f}_j = -\frac{1}{r_{ij}}\left[\frac{\partial}{\partial r_{ij}}U(r_{ij})\right]\underline{r}_{ij}, \tag{2.12}$$

The force $\underline{f}_i$ acting on atom i is the negative of this.

The contribution to be added to the atomic virial is given by

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j, \tag{2.13}$$

with only *one* such contribution from each bond.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^\alpha f_j^\beta, \tag{2.14}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived in this way is symmetric.

In DL_POLY Classic bond forces are handled by the routine BNDFRC.

[1] Note: some DL_POLY Classic routines may use the convention that $\underline{r_{ij}} = \underline{r}_i - \underline{r}_j$. Nobody's perfect.

2.2.2 Distance Restraints

In DL_POLY Classic distance restraints, in which the separation between two atoms, is maintained around some preset value r_0 is handled as a special case of bond potentials. As a consequence distance restraints may be applied only between atoms in the same molecule. Unlike with application of the "pure" bond potentials, the electrostatic and van der Waals interactions between the pair of atoms are still evaluated when distance restraints are applied. All the potential forms of the previous section are as avaliable distance restraints, although they have different key words:

1. Harmonic potential: (**-hrm**)
2. Morse potential: (**-mrs**)
3. 12-6 potential bond: (**-126**)
4. Restrained harmonic: (**-rhm**)
5. Quartic potential: (**-qur**)
6. Buckingham potential: (**-bck**)
7. FENE potential: (**-fen**)
8. Coulombic bond: (**-cou**)

In DL_POLY Classic distance restraints are handled by the routine BNDFRC.

2.2.3 Valence Angle Potentials

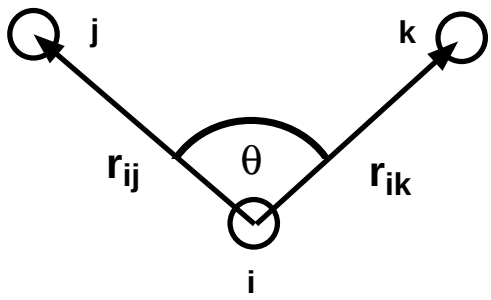

Figure 2.2: The valence angle and associated vectors

The valence angle potentials describe the bond bending terms between the specified atoms. They should not be confused with the three body potentials described later, which are defined by atom types rather than indices.

1. Harmonic: (**harm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2; \tag{2.15}$$
2. Quartic: (**quar**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 + \frac{k'}{3}(\theta_{jik} - \theta_0)^3 + \frac{k''}{4}(\theta_{jik} - \theta_0)^4; \tag{2.16}$$

3. Truncated harmonic: (**thrm**)

$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]; \tag{2.17}$$

4. Screened harmonic: (**shrm**)

$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \tag{2.18}$$

5. Screened Vessal[28]: (**bvs1**)

$$\begin{aligned} U(\theta_{jik}) &= \frac{k}{8(\theta_{jik} - \pi)^2} \left\{ \left[(\theta_0 - \pi)^2 - (\theta_{jik} - \pi)^2 \right]^2 \right\} \\ &\quad \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \end{aligned} \tag{2.19}$$

6. Truncated Vessal[29]: (**bvs2**)

$$\begin{aligned} U(\theta_{jik}) &= k[\theta_{jik}^a(\theta_{jik} - \theta_0)^2(\theta_{jik} + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1} \\ &\quad (\theta_{jik} - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]. \end{aligned} \tag{2.20}$$

7. Harmonic cosine: (**hcos**)

$$U(\theta_{jik}) = \frac{k}{2}(cos(\theta_{jik}) - cos(\theta_0))^2 \tag{2.21}$$

8. Cosine: (**cos**)

$$U(\theta_{jik}) = A[1 + cos(m\theta_{jik} - \delta)] \tag{2.22}$$

9. MM3 stretch-bend: (**mmsb**)

$$U(\theta_{jik}) = A(\theta_{jik} - \theta_0)(r_{ij} - r_{ij}^o)(r_{ik} - r_{ik}^o) \tag{2.23}$$

10. Compass stretch-stretch: (**stst**)

$$U_{jik} = A(r_{ij} - r_{ij}^o)(r_{ik} - r_{ik}^o) \tag{2.24}$$

11. Compass stretch-bend: (**stbe**)

$$U(\theta_{jik}) = A(\theta_{jik} - \theta_0)(r_{ij} - r_{ij}^o) \tag{2.25}$$

12. Compass all terms: (**cmps**)

$$\begin{aligned} U(\theta_{jik}) &= A(r_{ij} - r_{ij}^o)(r_{ik} - r_{ik}^o) + \\ &\quad (\theta_{jik} - \theta_0)(B(r_{ij} - r_{ij}^o) + C(r_{ik} - r_{ik}^o)) \end{aligned} \tag{2.26}$$

In these formulae θ_{jik} is the angle between bond vectors $\underline{r}_{ij}$ and $\underline{r}_{ik}$:

$$\theta_{jik} = cos^{-1}\left\{ \frac{\underline{r}_{ij} \cdot \underline{r}_{ik}}{r_{ij} r_{ik}} \right\} \tag{2.27}$$

In DL_POLY Classic the most general form for the valence angle potentials can be written as:

$$U(\theta_{jik}, r_{ij}, r_{ik}) = A(\theta_{jik})S(r_{ij})S(r_{ik}) \tag{2.28}$$

where $A(\theta)$ is a purely angular function and $S(r)$ is a screening or truncation function. All the function arguments are scalars. With this reduction the force on an atom derived from the valence angle potential is given by:

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha} U(\theta_{jik}, r_{ij}, r_{ik}), \tag{2.29}$$

with atomic label ℓ being one of i, j, k and α indicating the x, y, z component. The derivative is

$$\begin{aligned} -\frac{\partial}{\partial r_\ell^\alpha} U(\theta_{jik}, r_{ij}, r_{ik}) &= -S(r_{ij})S(r_{ik})\frac{\partial}{\partial r_\ell^\alpha} A(\theta_{jik}) \\ &\quad -A(\theta_{jik})S(r_{ik})(\delta_{\ell j} - \delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}}\frac{\partial}{\partial r_{ij}} S(r_{ij}) \\ &\quad -A(\theta_{jik})S(r_{ij})(\delta_{\ell k} - \delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ik}}\frac{\partial}{\partial r_{ik}} S(r_{ik}), \end{aligned} \tag{2.30}$$

with $\delta_{ab} = 1$ if $a = b$ and $\delta_{ab} = 0$ if $a \neq b$. In the absence of screening terms $S(r)$, this formula reduces to:

$$-\frac{\partial}{\partial r_\ell^\alpha} U(\theta_{jik}, r_{ij}, r_{ik}) = -\frac{\partial}{\partial r_\ell^\alpha} A(\theta_{jik}) \tag{2.31}$$

The derivative of the angular function is

$$-\frac{\partial}{\partial r_\ell^\alpha} A(\theta_{jik}) = \left\{ \frac{1}{\sin(\theta_{jik})} \right\} \frac{\partial}{\partial \theta_{jik}} A(\theta_{jik}) \frac{\partial}{\partial r_\ell^\alpha} \left\{ \frac{\underline{r}_{ij} \cdot \underline{r}_{ik}}{r_{ij} r_{ik}} \right\}, \tag{2.32}$$

with

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} \left\{ \frac{\underline{r}_{ij} \cdot \underline{r}_{ik}}{r_{ij} r_{ik}} \right\} &= (\delta_{\ell j} - \delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ij} r_{ik}} + (\delta_{\ell k} - \delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij} r_{ik}} - \\ &\quad \cos(\theta_{jik}) \left\{ (\delta_{\ell j} - \delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}^2} + (\delta_{\ell k} - \delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ik}^2} \right\} \end{aligned} \tag{2.33}$$

The atomic forces are then completely specified by the derivatives of the particular functions $A(\theta)$ and $S(r)$.

The contribution to be added to the atomic virial is given by

$$\mathcal{W} = -(\underline{r}_{ij} \cdot \underline{f}_j + \underline{r}_{ik} \cdot \underline{f}_k) \tag{2.34}$$

It is worth noting that in the absence of screening terms S(r), the virial is zero [30].

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^\alpha f_j^\beta + r_{ik}^\alpha f_k^\beta \tag{2.35}$$

and the stress tensor is symmetric.

In DL_POLY Classic valence forces are handled by the routine ANGFRC.

2.2.4 Angular Restraints

In DL_POLY Classic angle restraints, in which the angle subtended by a triplet of atoms, is maintained around some preset value θ_0 is handled as a special case of angle potentials. As a consequence angle restraints may be applied only between atoms in the same molecule. Unlike with application of the "pure" angle potentials, the electrostatic and van der Waals interactions between the pair of atoms are still evaluated when distance restraints are applied. All the potential forms of the previous section are available as angular restraints, although they have different key words:

1. Harmonic: (**-hrm**)
2. Quartic: (**-qur**)
3. Truncated harmonic: (**-thm**)
4. Screened harmonic: (**-shm**)
5. Screened Vessal[28]: (**-bv1**)
6. Truncated Vessal[29]: (**-bv2**)
7. Harmonic cosine: (**-hcs**)
8. Cosine : (**-cos**)
9. MM3 stretch-bend: (**-msb**)
10. Compass stretch-stretch (**-sts**)
11. Compass stretch-bend (**-stb**)
12. Compass all terms (**-cmp**)

In DL_POLY Classic angular restraints are handled by the routine ANGFRC.

2.2.5 Dihedral Angle Potentials

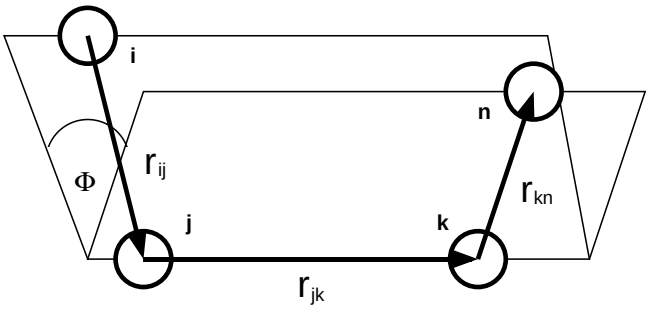

Figure 2.3: The dihedral angle and associated vectors

The dihedral angle potentials describe the interaction arising from torsional forces in molecules. (They are sometimes referred to as torsion potentials.) They require the specification of four atomic positions. The potential functions available in DL_POLY Classic are as follows.

1. Cosine potential: (**cos**)
$$U(\phi_{ijkn}) = A\left[1 + \cos(m\phi_{ijkn} - \delta)\right] \tag{2.36}$$
2. Harmonic: (**harm**)
$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.37}$$
3. Harmonic cosine: (**hcos**)
$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.38}$$

4. Triple cosine: (**cos3**)

$$U(\phi) = \frac{1}{2}A_1(1 + cos(\phi)) + \frac{1}{2}A_2(1 - cos(2\phi)) + \frac{1}{2}A_3(1 + cos(3\phi)) \tag{2.39}$$

5. Ryckaert-Bellemans hydrocarbon potential: (**ryck**)

$$U(\phi_{ijkn}) = A(a_0 + \sum_{i=1}^{5}(a_i cos^i(\phi)) \tag{2.40}$$

6. Ryckaert-Bellemans fluorinated potential: (**rbf**)

$$U(\phi_{ijkn}) = B(b_0 + \sum_{i=1}^{5}(b_i cos^i(\phi)) \tag{2.41}$$

7. OPLS angle potential

$$U(\phi_{ijkn}) = a_0 + 0.5 * (a_1(1 + cos(\phi)) + a_2(1 - cos(2\phi)) + a_3(1 + cos(3\phi))) \tag{2.42}$$

In these formulae ϕ_{ijkn} is the dihedral angle defined by

$$\phi_{ijkn} = \cos^{-1}\{B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn})\}, \tag{2.43}$$

with

$$B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}) = \left\{ \frac{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})}{|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|} \right\}. \tag{2.44}$$

With this definition, the sign of the dihedral angle is positive if the vector product $(\underline{r}_{ij} \times \underline{r}_{jk}) \times (\underline{r}_{jk} \times \underline{r}_{kn})$ is in the same direction as the bond vector $\underline{r}_{jk}$ and negative if in the opposite direction.

The force on an atom arising from the dihedral potential is given by

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha} U(\phi_{ijkn}), \tag{2.45}$$

with ℓ being one of i, j, k, n and α one of x, y, z. This may be expanded into

$$-\frac{\partial}{\partial r_\ell^\alpha} U(\phi_{ijkn}) = \left\{ \frac{1}{\sin(\phi_{ijkn})} \right\} \frac{\partial}{\partial \phi_{ijkn}} U(\phi_{ijkn}) \frac{\partial}{\partial r_\ell^\alpha} B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}). \tag{2.46}$$

The derivative of the function $B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn})$ is

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}) = & \frac{1}{|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|} \frac{\partial}{\partial r_\ell^\alpha} \{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})\} \\ & - \frac{\cos(\phi_{ijkn})}{2} \left\{ \frac{1}{|\underline{r}_{ij} \times \underline{r}_{jk}|^2} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{ij} \times \underline{r}_{jk}|^2 + \frac{1}{|\underline{r}_{jk} \times \underline{r}_{kn}|^2} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{jk} \times \underline{r}_{kn}|^2 \right\}, \end{aligned} \tag{2.47}$$

with

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} \{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})\} = & \; r_{ij}^\alpha ([\underline{r}_{jk}\underline{r}_{jk}]_\alpha (\delta_{\ell k} - \delta_{\ell n}) + [\underline{r}_{jk}\underline{r}_{kn}]_\alpha (\delta_{\ell k} - \delta_{\ell j})) + \\ & r_{jk}^\alpha ([\underline{r}_{ij}\underline{r}_{jk}]_\alpha (\delta_{\ell n} - \delta_{\ell k}) + [\underline{r}_{jk}\underline{r}_{kn}]_\alpha (\delta_{\ell j} - \delta_{\ell i})) + \\ & r_{kn}^\alpha ([\underline{r}_{ij}\underline{r}_{jk}]_\alpha (\delta_{\ell k} - \delta_{\ell j}) + [\underline{r}_{jk}\underline{r}_{jk}]_\alpha (\delta_{\ell i} - \delta_{\ell j})) + \\ & 2 r_{jk}^\alpha [\underline{r}_{ij}\underline{r}_{kn}]_\alpha (\delta_{\ell j} - \delta_{\ell k}), \end{aligned} \tag{2.48}$$

$$\frac{\partial}{\partial r_\ell^\alpha}|\underline{r}_{ij}\times\underline{r}_{jk}|^2 = 2r_{ij}^\alpha([\underline{r}_{jk}\underline{r}_{jk}]_\alpha(\delta_{\ell j}-\delta_{\ell i})+[\underline{r}_{ij}\underline{r}_{jk}]_\alpha(\delta_{\ell j}-\delta_{\ell k}))+ 2r_{jk}^\alpha([\underline{r}_{ij}\underline{r}_{ij}]_\alpha(\delta_{\ell k}-\delta_{\ell j})+[\underline{r}_{ij}\underline{r}_{jk}]_\alpha(\delta_{\ell i}-\delta_{\ell j})), \quad (2.49)$$

$$\frac{\partial}{\partial r_\ell^\alpha}|\underline{r}_{jk}\times\underline{r}_{kn}|^2 = 2r_{kn}^\alpha([\underline{r}_{jk}\underline{r}_{jk}]_\alpha(\delta_{\ell n}-\delta_{\ell k})+[\underline{r}_{jk}\underline{r}_{kn}]_\alpha(\delta_{\ell j}-\delta_{\ell k}))+ 2r_{jk}^\alpha([\underline{r}_{kn}\underline{r}_{kn}]_\alpha(\delta_{\ell k}-\delta_{\ell j})+[\underline{r}_{jk}\underline{r}_{kn}]_\alpha(\delta_{\ell k}-\delta_{\ell n})). \quad (2.50)$$

Where we have used the the following definition:

$$[\underline{a}\ \underline{b}]_\alpha=\sum_\beta(1-\delta_{\alpha\beta})a^\beta b^\beta. \quad (2.51)$$

Formally, the contribution to be added to the atomic virial is given by

$$\mathcal{W}=-\sum_{i=1}^4\underline{r}_i\cdot\underline{f}_i \quad (2.52)$$

However it is possible to show (by tedious algebra using the above formulae, or more elegantly by thermodynamic arguments [30],) that the dihedral makes *no* contribution to the atomic virial.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^\alpha p_i^\beta+r_{jk}^\alpha p_{jk}^\beta+r_{kn}^\alpha p_n^\beta -\frac{\cos(\phi_{ijkn})}{2}\left\{r_{ij}^\alpha g_i^\beta+r_{jk}^\alpha g_k^\beta+r_{jk}^\alpha h_j^\beta+r_{kn}^\alpha h_n^\beta\right\}, \quad (2.53)$$

with

$$p_i^\alpha = (r_{jk}^\alpha[\underline{r}_{jk}\underline{r}_{kn}]_\alpha-r_{kn}^\alpha[\underline{r}_{jk}\underline{r}_{jk}]_\alpha)/(|\underline{r}_{ij}\times\underline{r}_{jk}||\underline{r}_{jk}\times\underline{r}_{kn}|) \quad (2.54)$$

$$p_n^\alpha = (r_{jk}^\alpha[\underline{r}_{ij}\underline{r}_{jk}]_\alpha-r_{ij}^\alpha[\underline{r}_{jk}\underline{r}_{jk}]_\alpha)/(|\underline{r}_{ij}\times\underline{r}_{jk}||\underline{r}_{jk}\times\underline{r}_{kn}|) \quad (2.55)$$

$$p_{jk}^\alpha = (r_{ij}^\alpha[\underline{r}_{jk}\underline{r}_{kn}]_\alpha+r_{kn}^\alpha[\underline{r}_{ij}\underline{r}_{jk}]_\alpha-2r_{jk}^\alpha[\underline{r}_{ij}\underline{r}_{kn}]_\alpha)/(|\underline{r}_{ij}\times\underline{r}_{jk}||\underline{r}_{jk}\times\underline{r}_{kn}|) \quad (2.56)$$

$$g_i^\alpha = 2(r_{ij}^\alpha[\underline{r}_{jk}\underline{r}_{jk}]_\alpha-r_{jk}^\alpha[\underline{r}_{ij}\underline{r}_{jk}]_\alpha)/|\underline{r}_{ij}\times\underline{r}_{jk}|^2 \quad (2.57)$$

$$g_k^\alpha = 2(r_{jk}^\alpha[\underline{r}_{ij}\underline{r}_{ij}]_\alpha-r_{ij}^\alpha[\underline{r}_{ij}\underline{r}_{jk}]_\alpha)/|\underline{r}_{ij}\times\underline{r}_{jk}|^2 \quad (2.58)$$

$$h_j^\alpha = 2(r_{jk}^\alpha[\underline{r}_{kn}\underline{r}_{kn}]_\alpha-r_{kn}^\alpha[\underline{r}_{jk}\underline{r}_{kn}]_\alpha)/|\underline{r}_{jk}\times\underline{r}_{kn}|^2 \quad (2.59)$$

$$h_n^\alpha = 2(r_{kn}^\alpha[\underline{r}_{kn}\underline{r}_{kn}]_\alpha-r_{jk}^\alpha[\underline{r}_{jk}\underline{r}_{kn}]_\alpha)/|\underline{r}_{jk}\times\underline{r}_{kn}|^2 \quad (2.60)$$

The sum of the diagonal elements of the stress tensor is zero (since the virial is zero) and the matrix is symmetric.

Lastly, it should be noted that the above description does not take into account the possible inclusion of distance-dependent 1-4 interactions, as permitted by some force fields. Such interactions are permissible in DL_POLY Classic and are described in the section on pair potentials below. DL_POLY Classic also permits scaling of the 1-4 interactions by a numerical factor. 1-4 interactions do, of course, contribute to the atomic virial.

In DL_POLY Classic dihedral forces are handled by the routine DIHFRC.

2.2.6 Improper Dihedral Angle Potentials

Improper dihedrals are used to restrict the geometry of molecules and as such need not have a simple relation to conventional chemical bonding. DL_POLY Classic makes no distinction between

dihedral angle functions and improper dihedrals (both are calculated by the same subroutines) and all the comments made in the preceeding section apply.

An important example of the use of the improper dihedral is to conserve the structure of chiral centres in molecules modelled by united-atom centres. For example α-amino acids such as alanine ($CH_3CH(NH_2)COOH$), in which it is common to represent the CH_3 and CH groups as single centres. Conservation of the chirality of the α carbon is achieved by defining a harmonic improper dihedral angle potential with an equilibrium angle of 35.264^o. The angle is defined by vectors $\underline{r}_{12}$, $\underline{r}_{23}$ and $\underline{r}_{34}$, where the atoms 1,2,3 and 4 are shown in the following figure. The figure defines the D and L enantiomers consistent with the international (IUPAC) convention. When defining the dihedral, the atom indices are entered in DL_POLY Classic in the order 1-2-3-4.

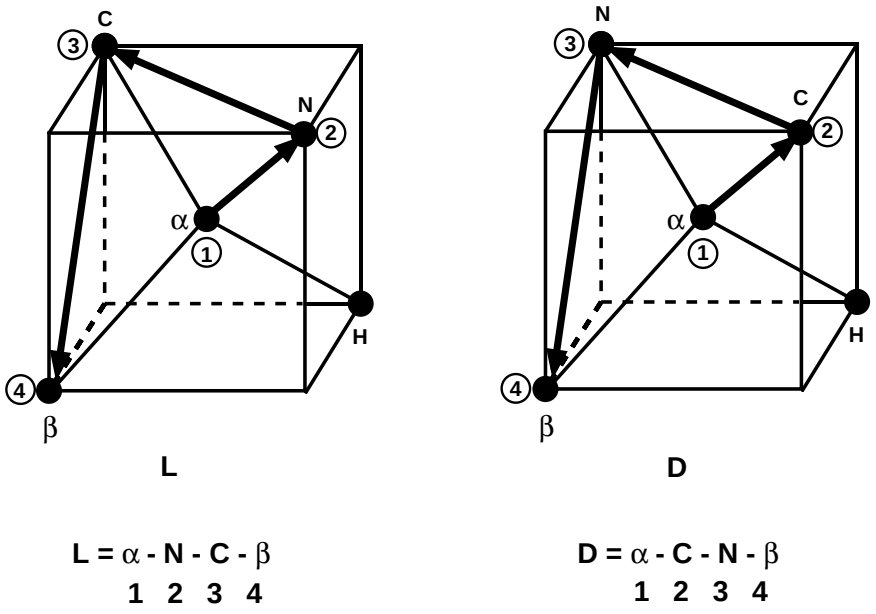

Figure 2.4: The L and D enantiomers and defining vectors

In DL_POLY Classic improper dihedral forces are handled by the routine DIHFRC.

2.2.7 Inversion Angle Potentials

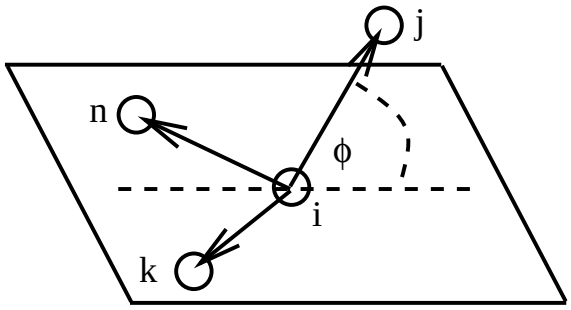

Figure 2.5: The inversion angle and associated vectors

The inversion angle potentials describe the interaction arising from a particular geometry of three atoms around a central atom. The best known example of this is the arrangement of hydrogen atoms around nitrogen in ammonia to form a trigonal pyramid. The hydrogens can 'flip' like an inverting umbrella to an alternative structure, which in this case is identical, but in principle causes a change in chirality. The force restraining the ammonia to one structure can be described as an inversion potential (though it is usually augmented by valence angle potentials also). The inversion angle is defined in the figure above - **note that the inversion angle potential is a sum of the three possible inversion angle terms.** It resembles a dihedral potential in that it requires the specification of four atomic positions.

The potential functions available in DL_POLY Classic are as follows.

1. Harmonic: (**harm**)
$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.61}$$

2. Harmonic cosine: (**hcos**)
$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.62}$$

3. Planar potential: (**plan**)
$$U(\phi_{ijkn}) = A\,[1 - \cos(\phi_{ijkn})] \tag{2.63}$$

In these formulae ϕ_{ijkn} is the inversion angle defined by

$$\phi_{ijkn} = \cos^{-1}\left\{\frac{\underline{r}_{ij}\cdot\underline{w}_{kn}}{r_{ij}w_{kn}}\right\}, \tag{2.64}$$

with

$$\underline{w}_{kn} = (\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{\underline{u}}_{kn} + (\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{\underline{v}}_{kn} \tag{2.65}$$

and the unit vectors

$$\begin{aligned}\hat{\underline{u}}_{kn} &= (\hat{\underline{r}}_{ik} + \hat{\underline{r}}_{in})/|\hat{\underline{r}}_{ik} + \hat{\underline{r}}_{in}| \\ \hat{\underline{v}}_{kn} &= (\hat{\underline{r}}_{ik} - \hat{\underline{r}}_{in})/|\hat{\underline{r}}_{ik} - \hat{\underline{r}}_{in}|.\end{aligned} \tag{2.66}$$

As usual, $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$ *etc.* and the hat $\hat{\underline{r}}$ indicates a *unit* vector in the direction of $\underline{r}$. The total inversion potential requires the calculation of three such angles, the formula being derived from the above using the cyclic permutation of the indices $j \to k \to n \to j$ *etc.*

Equivalently, the angle ϕ_{ijkn} may be written as

$$\phi_{ijkn} = \cos^{-1}\left\{\frac{[(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})^2 + (\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})^2]^{1/2}}{r_{ij}}\right\} \tag{2.67}$$

Formally, the force on an atom arising from the inversion potential is given by

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha}U(\phi_{ijkn}), \tag{2.68}$$

with ℓ being one of i, j, k, n and α one of x, y, z. This may be expanded into

$$\begin{aligned}-\frac{\partial}{\partial r_\ell^\alpha}U(\phi_{ijkn}) &= \left\{\frac{1}{\sin(\phi_{ijkn})}\right\}\frac{\partial}{\partial\phi_{ijkn}}U(\phi_{ijkn})\times \\ &\quad \frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{[(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})^2 + (\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})^2]^{1/2}}{r_{ij}}\right\}.\end{aligned} \tag{2.69}$$

Following through the (extremely tedious!) differentiation gives the result:

$$
\begin{aligned}
f_\ell^\alpha \;=\; & \left\{\frac{1}{\sin(\phi_{ijkn})}\right\}\frac{\partial}{\partial\phi_{ijkn}}U(\phi_{ijkn})\times \\
& \left\{-(\delta_{\ell j}-\delta_{\ell i})\frac{cos(\phi_{ijkn})}{r_{ij}^2}r_{ij}^\alpha+\frac{1}{r_{ij}w_{kn}}\left[(\delta_{\ell j}-\delta_{\ell i})\{(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha+(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha\}\right.\right. \\
& +(\delta_{\ell k}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn}}{u_{kn}r_{ik}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{ik}-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})(\underline{r}_{ik}\cdot\hat{\underline{u}}_{kn}))\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \\
& +(\delta_{\ell k}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn}}{v_{kn}r_{ik}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{ik}-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})(\underline{r}_{ik}\cdot\hat{\underline{v}}_{kn}))\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \\
& +(\delta_{\ell n}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn}}{u_{kn}r_{in}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{in}-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})(\underline{r}_{in}\cdot\hat{\underline{u}}_{kn}))\frac{r_{in}^\alpha}{r_{in}^2}\right\} \\
& \left.\left.-(\delta_{\ell n}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn}}{v_{kn}r_{in}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{in}-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})(\underline{r}_{in}\cdot\hat{\underline{v}}_{kn}))\frac{r_{in}^\alpha}{r_{in}^2}\right\}\right]\right\}
\end{aligned}
\tag{2.70}
$$

This general formula applies to all atoms $\ell = i, j, k, n$. It must be remembered however, that these formulae apply to just one of the three contributing terms (i.e. one angle ϕ) of the full inversion potential: specifically the inversion angle pertaining to the out-of-plane vector $\underline{r}_{ij}$. The contributions arising from the other vectors $\underline{r}_{ik}$ and $\underline{r}_{in}$ are obtained by the cyclic permutation of the indices in the manner described above. All these force contributions must be added to the final atomic forces.

Formally, the contribution to be added to the atomic virial is given by

$$\mathcal{W} = -\sum_{i=1}^{4}\underline{r}_i\cdot\underline{f}_i \tag{2.71}$$

However it is possible to show by thermodynamic arguments (*cf* [30],) or simply from the fact that the sum of forces on atoms j,k and n is equal and opposite to the force on atom i, that the inversion potential makes *no* contribution to the atomic virial.

If the force components f_ℓ^α for atoms $\ell = i, j, k, n$ are calculated using the above formulae, it is easily seen that the contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^\alpha f_j^\beta + r_{ik}^\alpha f_k^\beta + r_{in}^\alpha f_n^\beta \tag{2.72}$$

The sum of the diagonal elements of the stress tensor is zero (since the virial is zero) and the matrix is symmetric.

In DL_POLY Classic inversion forces are handled by the routine INVFRC.

2.2.8 The Calcite Four-Body Potential

This potential [31] is designed to help maintain the planar structure of the carbonate anion $[CO_3]^{2-}$ in a similar manner to the planar inversion potential described above. However it is *not* an angular potential. It is dependent on the perpendicular displacement (u) of an atom a from a plane defined by three other atoms b, c, and d (see figure 2.6) and has the form

$$U_{abcd}(u) = Au^2 + Bu^4 \tag{2.73}$$

Where the displacement u is given by

$$u = \frac{\underline{r}_{ab}\cdot\underline{r}_{bc}\times\underline{r}_{bd}}{|\underline{r}_{bc}\times\underline{r}_{bd}|}. \tag{2.74}$$

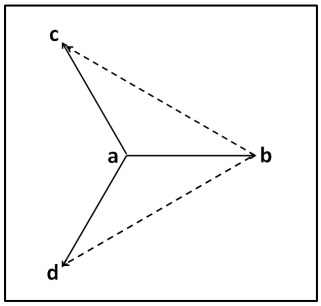

Figure 2.6: The vectors of the calcite potential

Vectors $\underline{r}_{ab}$,$\underline{r}_{ac}$ and $\underline{r}_{ad}$ define bonds between the central atom a and the peripheral atoms b, c and d. Vectors $\underline{r}_{bc}$ and $\underline{r}_{bd}$ define the plane and are related to the bond vectors by:

$$\begin{aligned} \underline{r}_{bc} &= \underline{r}_{ac} - \underline{r}_{ab} \\ \underline{r}_{bd} &= \underline{r}_{ad} - \underline{r}_{ab}. \end{aligned} \tag{2.75}$$

It what follows it is convenient to define the vector product appearing in both the numerator and denominator of equation (2.74) as the vector $\underline{w}_{cd}$ *vis.*

$$\underline{w}_{cd} = \underline{r}_{bc} \times \underline{r}_{bd} \tag{2.76}$$

We also define the quantity $\gamma(u)$ as

$$\gamma(u) = -(2Au + 4Bu^3). \tag{2.77}$$

The forces on the individual atoms due to the calcite potential are then given by

$$\begin{aligned} \underline{f}_a &= -\gamma(u)\hat{\underline{w}}_{cd} \\ \underline{f}_c &= \underline{r}_{bd} \times (\underline{r}_{ab} - u\hat{\underline{w}}_{cd})\gamma(u)/w_{cd} \\ \underline{f}_d &= -\underline{r}_{bc} \times (\underline{r}_{ab} - u\hat{\underline{w}}_{cd})\gamma(u)/w_{cd} \\ \underline{f}_b &= -(\underline{f}_a + \underline{f}_c + \underline{f}_d), \end{aligned} \tag{2.78}$$

where $w_{cd} = |\underline{w}_{cd}|$ and $\hat{\underline{w}}_{cd} = \underline{w}_{cd}/w_{cd}$. The virial contribution $\psi_{abcd}(u)$ is given by

$$\psi_{abcd}(u) = 2Au^2 + 4Bu^4 \tag{2.79}$$

and the stress tensor contribution $\sigma^{\alpha\beta}_{abcd}(u)$ by

$$\sigma^{\alpha\beta}_{abcd}(u) = \frac{u\gamma(u)}{w^2_{cd}} w^{\alpha}_{cd} w^{\beta}_{cd}. \tag{2.80}$$

In DL_POLY Classic the calcite forces are handled by the routine INVFRC, which is a convenient *intramolecular* four-body force routine. However it is manifestly *not* an inversion potential as such.

2.2.9 Tethering Forces

DL_POLY Classic also allows atomic sites to be tethered to a fixed point in space, $\underline{r}_0$ taken as their position at the beginning of the simulation. This is also known as position restraining. The

specification, which comes as part of the molecular description, requires a tether potential type and the associated interaction parameters.

Note, firstly, that application of tethering potentials means that momentum will no longer be a conserved quantity of the simulation. Secondly, in constant pressure simulations, where the MD cell changes size or shape, the reference position is scaled with the cell vectors.

The potential functions available in DL_POLY Classic are as follows, in each case r_{i0} is the distance of the atom from its position at $t = 0$:

1. harmonic potential: (**harm**)
$$U(r_{i0}) = \frac{1}{2}k(r_{i0})^2; \tag{2.81}$$

2. restrained harmonic :(**rhrm**)
$$U(r_{i0}) = \frac{1}{2}k(r_{i0})^2 \qquad r_{i0} \leq r_c; \tag{2.82}$$
$$U(r_{i0}) = \frac{1}{2}kr_c^2 + kr_c(r_{i0} - r_c) \qquad r_{i0} > r_c; \tag{2.83}$$

3. Quartic potential: (**quar**)
$$U(r_{i0}) = \frac{k}{2}(r_{i0})^2 + \frac{k'}{3}(r_{i0})^3 + \frac{k''}{4}(r_{i0})^4. \tag{2.84}$$

The force on the atom i arising from a tether potential is obtained using the general formula:

$$\underline{f}_i = -\frac{1}{r_{i0}}\left[\frac{\partial}{\partial r_{i0}}U(r_{i0})\right]\underline{r}_{i0}, \tag{2.85}$$

The contribution to be added to the atomic virial is given by

$$\mathcal{W} = \underline{r}_{i0} \cdot \underline{f}_i, \tag{2.86}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = -r_{i0}^{\alpha} f_i^{\beta}, \tag{2.87}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived in this way is symmetric.

In DL_POLY Classic bond forces are handled by the routine TETHFRC.

2.2.10 Frozen Atoms

DL_POLY Classic also allows atoms to be completely immobilised (*i.e.* "frozen" at a fixed point in the MD cell). This is achieved by setting all forces and velocities associated with that atom to zero during each MD timestep. Frozen atoms are signalled by assigning an atom a non-zero value for the freeze parameter in the FIELD file. DL_POLY Classic does not calculate contributions to the virial or the stress tensor arising from the constraints required to freeze atomic positions. In DL_POLY Classic the frozen atom option cannot be used for sites in a rigid body. As with the tethering potential, the reference position is scaled with the cell vectors in constant pressure simulations.

In DL_POLY Classic the frozen atom option is handled by the subroutine FREEZE.

2.3 The Intermolecular Potential Functions

In this section we outline the pair-body, three-body and four-body potential functions available in DL_POLY Classic. An important distinction between these and intramolecular (bond) forces in DL_POLY Classic is that they are specified by *atom types* rather than atom indices.

2.3.1 Short Ranged (van der Waals) Potentials

The short ranged pair forces available in DL_POLY Classic are as follows.

1. 12 - 6 potential: (**12-6**)
$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{6}}\right); \tag{2.88}$$

2. Lennard-Jones: (**lj**)
$$U(r_{ij}) = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]; \tag{2.89}$$

3. n - m potential [32]: (**nm**)
$$U(r_{ij}) = \frac{E_o}{(n-m)}\left[m\left(\frac{r_o}{r_{ij}}\right)^{n} - n\left(\frac{r_o}{r_{ij}}\right)^{m}\right]; \tag{2.90}$$

4. Buckingham potential: (**buck**)
$$U(r_{ij}) = A\ \exp\left(-\frac{r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^{6}}; \tag{2.91}$$

5. Born-Huggins-Meyer potential: (**bhm**)
$$U(r_{ij}) = A\ \exp[B(\sigma - r_{ij})] - \frac{C}{r_{ij}^{6}} - \frac{D}{r_{ij}^{8}}; \tag{2.92}$$

6. Hydrogen-bond (12 - 10) potential: (**hbnd**)
$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{10}}\right); \tag{2.93}$$

7. Shifted force n - m potential [32]: (**snm**)
$$\begin{aligned} U(r_{ij}) &= \frac{\alpha E_o}{(n-m)}\left[m\beta^n\left\{\left(\frac{r_o}{r_{ij}}\right)^{n} - \left(\frac{1}{\gamma}\right)^{n}\right\} - n\beta^m\left\{\left(\frac{r_o}{r_{ij}}\right)^{m} - \left(\frac{1}{\gamma}\right)^{m}\right\}\right] \\ &\quad + \frac{nm\alpha E_o}{(n-m)}\left(\frac{r_{ij} - \gamma r_o}{\gamma r_o}\right)\left\{\left(\frac{\beta}{\gamma}\right)^{n} - \left(\frac{\beta}{\gamma}\right)^{m}\right\} \end{aligned} \tag{2.94}$$

with

$$\gamma = \frac{r_{cut}}{r_o} \tag{2.95}$$

$$\beta = \gamma\left(\frac{\gamma^{m+1}-1}{\gamma^{n+1}-1}\right)^{\frac{1}{n-m}} \tag{2.96}$$

$$\alpha = \frac{(n-m)}{[n\beta^m(1+(m/\gamma-m-1)/\gamma^m) - m\beta^n(1+(n/\gamma-n-1)/\gamma^n)]} \tag{2.97}$$

This peculiar form has the advantage over the standard shifted n-m potential in that both E_o and r_0 (well depth and location of minimum) retain their original values after the shifting process.

8. Morse potential: (**mors**)

$$U(r_{ij}) = E_o[\{1 - \exp(-k(r_{ij} - r_o))\}^2 - 1]; \tag{2.98}$$

9. Shifted Weeks-Chandler-Anderson (WCA) potential [33]: (**wca**)

$$U(r_{ij}) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r_{ij}-\Delta}\right)^{12} - \left(\frac{\sigma}{r_{ij}-\Delta}\right)^{6}\right] + \epsilon & : \quad r_{ij} < 2^{\frac{1}{6}}\ \sigma + \Delta \\ 0 & : \quad r_{ij} \geq 2^{\frac{1}{6}}\ \sigma + \Delta \end{cases} \tag{2.99}$$

The WCA potential is the Lennard-Jones potential truncated at the position of the minimum and shifted to eliminate discontinuity (includes the effect of excluded volume). It is usually used in combination with the FENE (2.9) bond potential. This implementation allows for a radius shift of up to half a σ ($|\Delta| \leq 0.5\ \sigma$) with a default of zero ($\Delta_{default} = 0$).

10. Gaussian potential (**gaus**)

$$U(r_{ij}) = \sum_{n}^{3} A_n exp(-b_b r_{ij}^2) \tag{2.100}$$

Up to 3 Gaussian terms are permitted, unrequired terms have $A_n = 0$.

11. Tabulation: (**tab**). The potential is defined numerically only.

The parameters defining these potentials are supplied to DL_POLY Classic at run time (see the description of the FIELD file in section 4.1.3). Each atom type in the system is specified by a unique eight-character label defined by the user. The pair potential is then defined internally by the combination of two atom labels.

As well as the numerical parameters defining the potentials, DL_POLY Classic must also be provided with a cutoff radius r_{cut}, which sets a ranged limit on the computation of the interaction. Together with the parameters, the cutoff is used by the subroutine FORGEN (or FORGEN_RSQ) to construct an interpolation array `vvv` for the potential function over the ranged 0 to r_{cut}. A second array `ggg` is also calculated, which is related to the potential via the formula:

$$G(r_{ij}) = -r_{ij}\frac{\partial}{\partial r_{ij}}U(r_{ij}), \tag{2.101}$$

and is used in the calculation of the forces. Both arrays are tabulated in units of energy. The use of interpolation arrays, rather than the explicit formulae, makes the routines for calculating the potential energy and atomic forces very general, and enables the use of user defined pair potential functions. DL_POLY Classic also allows the user to read in the interpolation arrays directly from a file (see the description of the TABLE file (section 4.1.5). This is particularly useful if the pair potential function has no simple analytical description (e.g. spline potentials).

The force on an atom j derived from one of these potentials is formally calculated with the standard formula:

$$\underline{f}_j = -\frac{1}{r_{ij}}\left[\frac{\partial}{\partial r_{ij}}U(r_{ij})\right]\underline{r}_{ij}, \tag{2.102}$$

where $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. The force on atom i is the negative of this.

The contribution to be added to the atomic virial (for each pair interaction) is

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j. \tag{2.103}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.104}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived from the pair forces is symmetric.

Since the calculation of pair potentials assumes a spherical cutoff (r_{cut}) it is necessary to apply a *long ranged correction* to the system potential energy and virial. Explicit formulae are needed for each case and are derived as follows. For two atom types a and b, the correction for the potential energy is calculated via the integral

$$U_{corr}^{ab} = 2\pi \frac{N_a N_b}{V} \int_{r_{cut}}^{\infty} g_{ab}(r) U_{ab}(r) r^2 dr \tag{2.105}$$

where N_a, N_b are the numbers of atoms of types a and b, V is the system volume and $g_{ab}(r)$ and $U_{ab}(r)$ are the appropriate pair correlation function and pair potential respectively. It is usual to assume $g_{ab}(r) = 1$ for $r > r_{cut}$. DL_POLY Classic sometimes makes the additional assumption that the repulsive part of the short ranged potential is negligible beyond r_{cut}.

The correction for the system virial is

$$\mathcal{W}_{corr}^{ab} = -2\pi \frac{N_a N_b}{V} \int_{r_{cut}}^{\infty} g_{ab}(r) \frac{\partial}{\partial r} U_{ab}(r) r^3 dr, \tag{2.106}$$

where the same approximations are applied. Note that these formulae are based on the assumption that the system is reasonably isotropic beyond the cutoff.

In DL_POLY Classic the short ranged forces are calculated by one of the routines SRFRCE, SRFRCE_RSQ, and SRFRCENEU. The long ranged corrections are calculated by routine LRCORRECT. The calculation makes use of the Verlet neighbour list described above.

2.3.2 Three Body Potentials

The three-body potentials in DL_POLY Classic are mostly valence angle forms. (They are primarily included to permit simulation of amorphous materials e.g. silicate glasses.) However, these have been extended to include the Dreiding [7] hydrogen bond. The potential forms available are as follows.

1. Harmonic: (**harm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \tag{2.107}$$

2. Truncated harmonic: (**thrm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]; \tag{2.108}$$

3. Screened Harmonic: (**shrm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \tag{2.109}$$

4. Screened Vessal[28]: (**bvs1**)

$$U(\theta_{jik}) = \frac{k}{8(\theta_{jik}-\pi)^2}\left\{\left[(\theta_0-\pi)^2-(\theta_{jik}-\pi)^2\right]^2\right\} \exp[-(r_{ij}/\rho_1+r_{ik}/\rho_2)]; \quad (2.110)$$

5. Truncated Vessal[29]: (**bvs2**)

$$U(\theta_{jik}) = k[\theta^a_{jik}(\theta_{jik}-\theta_0)^2(\theta_{jik}+\theta_0-2\pi)^2-\frac{a}{2}\pi^{a-1} (\theta_{jik}-\theta_0)^2(\pi-\theta_0)^3]\exp[-(r^8_{ij}+r^8_{ik})/\rho^8]. \quad (2.111)$$

6. Dreiding hydrogen bond [7]: (**hbnd**)

$$U(\theta_{jik}) = D_{hb}cos^4(\theta_{jik})[5(R_{hb}/r_{jk})^{12}-6(R_{hb}/r_{jk})^{10}] \quad (2.112)$$

Note that for the hydrogen bond, the hydrogen atom *must* be the central atom. Several of these functions are identical to those appearing in the *intra*-molecular valenceangle descriptions above. There are significant differences in implementation however, arising from the fact that the three-body potentials are regarded as *inter*-molecular. Firstly, the atoms involved are defined by atom types, not specific indices. Secondly, there are *no* excluded atoms arising from the three body terms. (The inclusion of pair potentials may in fact be essential to maintain the structure of the system.)

The three body potentials are very short ranged, typically of order 3 $\mathring{A}$. This property, plus the fact that three body potentials scale as N^3, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [34].

The calculation of the forces, virial and stress tensor as described in the section valence angle potentials above.

DL_POLY Classic applies no long ranged corrections to the three body potentials. The three body forces are calculated by the routine THBFRC.

2.3.3 The Tersoff Covalent Potential

The Tersoff potential [5] is a special example of a density dependent potential, which has been designed to reproduce the properties of covalent bonding in systems containing carbon, silicon, germanium etc and alloys of these elements. A special feature of the potential is that it allows bond breaking and associated changes in bond hybridisation. The potential has 11 atomic and 2 bi-atomic parameters. The energy is modelled as a sum of pair-like interactions where, however, the coefficient of the attractive term in the pairlike potential (which plays the role of a bond order) depends on the local environment giving a many-body potential.

The form of the Tersoff potential is: (**ters**)

$$U_{ij} = f_C(r_{ij})\ [f_R(r_{ij}) - \gamma_{ij}\ f_A(r_{ij})], \quad (2.113)$$

where

$$f_R(r_{ij}) = A_{ij}\ \exp(-a_{ij}\ r_{ij})\ , \quad f_A(r_{ij}) = B_{ij}\ \exp(-b_{ij}\ r_{ij}) \quad (2.114)$$

$$f_C(r_{ij}) = \begin{cases} 1 & : \quad r_{ij} < R_{ij} \\ \frac{1}{2}+\frac{1}{2}\cos[\pi\ (r_{ij}-R_{ij})/(r_{ij}-R_{ij})] & : \quad R_{ij} < r_{ij} < S_{ij} \\ 0 & : \quad r_{ij} > S_{ij} \end{cases} \quad (2.115)$$

$$\gamma_{ij} = \chi_{ij}\ (1 + {\beta_i}^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i})^{-1/2\eta_i}\ ,\quad \mathcal{L}_{ij} = \sum_{k\neq i,j} f_C(r_{ik})\ \omega_{ik}\ g(\theta_{ijk})$$
$$g(\theta_{ijk}) = 1 + c_i^2/d_i^2 - c_i^2/[d_i^2 + (h_i - \cos\theta_{ijk})^2] \tag{2.116}$$

with further mixed parameters defined as

$$\begin{aligned} a_{ij} &= (a_i + a_j)/2 \quad , \quad b_{ij} = (b_i + b_j)/2 \\ A_{ij} &= (A_i A_j)^{1/2} \quad , \quad B_{ij} = (B_i B_j)^{1/2} \\ R_{ij} &= (R_i R_j)^{1/2} \quad , \quad S_{ij} = (S_i S_j)^{1/2} \quad . \end{aligned} \tag{2.117}$$

Here $i,\ j$ and k label the atoms in the system, r_{ij} is the length of the ij bond, and θ_{ijk} is the bond angle between bonds ij and ik. Single subscripted parameters (11), such as a_i and η_i, depend only on the type of atom.

The chemistry between different atom types is encapsulated in the two sets of bi-atomic parameters χ_{ij} and ω_{ij}:

$$\begin{aligned} \chi_{ii} &= 1 \quad , \quad \chi_{ij} = \chi_{ji} \\ \omega_{ii} &= 1 \quad , \quad \omega_{ij} = \omega_{ji} \quad , \end{aligned} \tag{2.118}$$

which define only one independent parameter for each pair of atom types. The χ parameter is used to strengthen or weaken the heteropolar bonds, relative to the value obtained by simple interpolation. The ω parameter is used to permit greater flexibility when dealing with more drastically different types of atoms.

The force on an atom ℓ derived from this potential is formally calculated with the formula:

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha} E_{\texttt{tersoff}} = \frac{1}{2}\sum_{i\neq j} -\frac{\partial}{\partial r_\ell^\alpha} U_{ij} \quad , \tag{2.119}$$

with atomic label ℓ being one of i, j, k and α indicating the x, y, z component. The derivative in the above formula expands into

$$-\frac{\partial U_{ij}}{\partial r_\ell^\alpha} = -\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_R(r_{ij}) + \gamma_{ij}\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_A(r_{ij}) + f_C(r_{ij}) f_A(r_{ij})\frac{\partial}{\partial r_\ell^\alpha}\gamma_{ij} \quad , \tag{2.120}$$

with the contributions from the first two terms being:

$$\begin{aligned} -\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_R(r_{ij}) &= -\left\{ f_C(r_{ij})\frac{\partial}{\partial r_{ij}} f_R(r_{ij}) + f_R(r_{ij})\frac{\partial}{\partial r_{ij}} f_C(r_{ij}) \right\} \times \\ &\quad \left\{ \delta_{j\ell}\frac{r_{i\ell}^\alpha}{r_{i\ell}} - \delta_{i\ell}\frac{r_{\ell j}^\alpha}{r_{\ell j}} \right\} \end{aligned} \tag{2.121}$$

$$\begin{aligned} \gamma_{ij}\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_A(r_{ij}) &= \gamma_{ij}\left\{ f_C(r_{ij})\frac{\partial}{\partial r_{ij}} f_A(r_{ij}) + f_A(r_{ij})\frac{\partial}{\partial r_{ij}} f_C(r_{ij}) \right\} \times \\ &\quad \left\{ \delta_{j\ell}\frac{r_{i\ell}^\alpha}{r_{i\ell}} - \delta_{i\ell}\frac{r_{\ell j}^\alpha}{r_{\ell j}} \right\} \quad , \end{aligned} \tag{2.122}$$

and from the third (angular) term:

$$\begin{aligned} f_C(r_{ij}) f_A(r_{ij})\frac{\partial}{\partial r_\ell^\alpha}\gamma_{ij} &= f_C(r_{ij}) f_A(r_{ij})\ \chi_{ij} \quad \times \\ &\quad \left(-\frac{1}{2}\right)\left(1 + {\beta_i}^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i}\right)^{-\frac{1}{2\eta_i}-1} {\beta_i}^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i - 1}\frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} \quad , \end{aligned} \tag{2.123}$$

where

$$\frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} = \frac{\partial}{\partial r_\ell^\alpha}\sum_{k\neq i,j}\omega_{ik}\ f_C(r_{ik})\ g(\theta_{ijk}) \quad . \tag{2.124}$$

The angular term can have three different contributions depending on the index of the particle participating in the interaction:

$$\ell = i \quad : \quad \frac{\partial}{\partial r_i^\alpha}\mathcal{L}_{ij} = \sum_{k\neq i,j}\omega_{ik}\left[g(\theta_{ijk})\frac{\partial}{\partial r_i^\alpha}f_C(r_{ik}) + f_C(r_{ik})\frac{\partial}{\partial r_i^\alpha}g(\theta_{ijk})\right] \tag{2.125}$$

$$\ell = j \quad : \quad \frac{\partial}{\partial r_j^\alpha}\mathcal{L}_{ij} = \sum_{k\neq i,j}\omega_{ik}\ f_C(r_{ik})\frac{\partial}{\partial r_j^\alpha}g(\theta_{ijk}) \tag{2.126}$$

$$\ell \neq i,j \quad : \quad \frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} = \omega_{i\ell}\left[g(\theta_{ij\ell})\frac{\partial}{\partial r_\ell^\alpha}f_C(r_{i\ell}) + f_C(r_{i\ell})\frac{\partial}{\partial r_\ell^\alpha}g(\theta_{ij\ell})\right] \quad . \tag{2.127}$$

The derivative of $g(\theta_{ijk})$ is worked out in the following manner:

$$\frac{\partial}{\partial r_\ell^\alpha}g(\theta_{ijk}) = \frac{\partial g(\theta_{ijk})}{\partial\theta_{ijk}}\ \frac{-1}{\sin\theta_{ijk}}\ \frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{\underline{r}_{ij}\cdot\underline{r}_{ik}}{r_{ij}\ r_{ik}}\right\} \quad , \tag{2.128}$$

where

$$\frac{\partial g(\theta_{ijk})}{\partial\theta_{ijk}} = \frac{2\ c_i^2(h_i-\cos\theta_{ijk})\ \sin\theta_{ijk}}{[d_i^2+(h_i-\cos\theta_{ijk})^2]^2} \tag{2.129}$$

$$\begin{aligned}\frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{\underline{r}_{ij}\cdot\underline{r}_{ik}}{r_{ij}r_{ik}}\right\} &= (\delta_{\ell j}-\delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ij}r_{ik}} + (\delta_{\ell k}-\delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}r_{ik}} - \\ &\quad \cos(\theta_{jik})\left\{(\delta_{\ell j}-\delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}^2} + (\delta_{\ell k}-\delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \quad .\end{aligned} \tag{2.130}$$

The contribution to be added to the atomic virial can be derived as

$$\mathcal{W} = 3V\frac{\partial E_{\texttt{tersoff}}}{\partial V} = \frac{3\ V}{2}\sum_{i\neq j}\frac{\partial U_{ij}}{\partial V} \tag{2.131}$$

$$\begin{aligned}\mathcal{W} = \frac{1}{2}\sum_i\sum_{j\neq i}\Bigg\{&\left[\frac{\partial}{\partial r_{ij}}f_C(r_{ij})f_R(r_{ij}) - \gamma_{ij}\frac{\partial}{\partial r_{ij}}f_C(r_{ij})f_A(r_{ij})\right]r_{ij} - \\ &\left(-\frac{1}{2}\right)f_C(r_{ij})f_A(r_{ij})\ \chi_{ij}\left(1+\beta_i^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i}\right)^{-\frac{1}{2\eta_i}-1}\beta_i^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i-1}\times \\ &\sum_{k\neq i,j}\omega_{ik}\ g(\theta_{ijk})\left[\frac{\partial}{\partial r_{ik}}f_C(r_{ik})\right]r_{ik}\quad\Bigg\}.\end{aligned} \tag{2.132}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = -r_i^\alpha f_i^\beta \quad , \tag{2.133}$$

where α and β indicate the x, y, z components. The stress tensor is symmetric.

Interpolation arrays, `vmbp` and `gmbp` (set up in subroutine TERGEN) - similar to those in van der Waals interactions 2.3.1 - are used in the calculation of the Tersoff forces, virial and stress.

The Tersoff potentials are very short ranged, typically of order 3 Å. This property, plus the fact that Tersoff potentials (two- and three-body contributions) scale as N^3, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [34].

DL_POLY Classic applies no long ranged corrections to the Tersoff potentials. In DL_POLY Classic Tersoff forces are handled by the routines TERSOFF, TERINT and TERSOFF3.

2.3.4 Four Body Potentials

The four-body potentials in DL_POLY Classic are entirely inversion angle forms, primarily included to permit simulation of amorphous materials (particularly borate glasses). The potential forms available in DL_POLY Classic are as follows.

1. Harmonic: (**harm**)
$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.134}$$
2. Harmonic cosine: (**hcos**)
$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.135}$$
3. Planar potential: (**plan**)
$$U(\phi_{ijkn}) = A[1 - \cos(\phi_{ijkn})] \tag{2.136}$$

These functions are identical to those appearing in the *intra*-molecular inversion angle descriptions above. There are significant differences in implementation however, arising from the fact that the four-body potentials are regarded as *inter*-molecular. Firstly, the atoms involved are defined by atom types, not specific indices. Secondly, there are *no* excluded atoms arising from the four-body terms. (The inclusion of other potentials, for example pair potentials, may in fact be essential to maintain the structure of the system.)

The four body potentials are very short ranged, typically of order 3 $\mathring{A}$. This property, plus the fact that four body potentials scale as N^4, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [34].

The calculation of the forces, virial and stress tensor described in the section on inversion angle potentials above.

DL_POLY Classic applies no long ranged corrections to the four body potentials. The four-body forces are calculated by the routine FBPFRC.

2.3.5 Metal Potentials

The metal potentials in DL_POLY Classic follow two similar but distinct formalisms. The first of these is the embedded atom model (EAM) [35, 36] and the second is the Finnis-Sinclair model (FSM) [3]. Both are density dependent potentials derived from density functional theory (DFT) and describe the bonding of a metal atom ultimately in terms of the local electronic density. They are suitable for calculating the properties of metals and metal alloys.

For single component metals the two approaches are the same. **However** they are subtly different in the way they are extended to handle alloys (see below). It follows that EAM and FSM potentials cannot be mixed in a single simulation. Furthermore, even for FSM potentials possessing different analytical forms there is no agreed procedure for mixing the parameters. The user is therefore strongly advised to be consistent in the choice of potential when modelling alloys.

The general form of the EAM and FSM potentials is [37]

$$U_{metal} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} V_{ij}(r_{ij}) + \sum_{i=1}^{N} F(\rho_i) \quad , \tag{2.137}$$

where $F(\rho_i)$ is a functional describing the energy of embedding an atom in the bulk density, ρ_i, which is defined as

$$\rho_i = \sum_{j=1, j\neq i}^{N} \rho_{ij}(r_{ij}) \quad . \tag{2.138}$$

It should be noted that the density is determined by the coordination number of the atom defined by *pairs* of atoms. This makes the metal potential dependent on the local density (environmental). $V_{ij}(r_{ij})$ is a pair potential incorporating repulsive electrostatic and overlap interactions. N is the number of interacting particles in the MD box.

The types of metal potentials available in DL_POLY Classic are as follows:

1. EAM potential: (**eam**) There are no explicit mathematical expressions for EAM potentials, so this potential type is read exclusively in the form of interpolation arrays from the TABEAM table file (as implemented in the METTAB routine - Section 4.1.6.) The rules for combining the potentials from different metals to handle alloys are different from the FSM class of potentials (see below).

2. Finnis-Sinclair potential [3]: (**fnsc**) The Finnis-Sinclair potential is explicitly analytical. It has the following form:

$$\begin{aligned} V_{ij}(r_{ij}) &= (r_{ij}-c)^2(c_0+c_1 r_{ij}+c_2 r_{ij}^2) \\ \rho_{ij}(r_{ij}) &= (r_{ij}-d)^2+\beta\frac{(r_{ij}-d)^3}{d} \\ F(\rho_i) &= -A\sqrt{\rho_i} \quad , \end{aligned} \tag{2.139}$$

   with parameters: c_0, c_1, c_2, c, A, d, β, both c and d are cutoffs. Since first being proposed a number of alternative analytical forms have been proposed, some of which are descibed below. The rules for combining different metal potentials to model alloys are different from the EAM potentials (see below).

3. Sutton-Chen potential [38, 39, 40]: (**stch**) The Sutton Chen potential is an analytical potential in the FSM class. It has the form:

$$\begin{aligned} V_{ij}(r_{ij}) &= \epsilon\left(\frac{a}{r_{ij}}\right)^n \\ \rho_{ij}(r_{ij}) &= \left(\frac{a}{r_{ij}}\right)^m \\ F(\rho_i) &= -c\epsilon\sqrt{\rho_i} \quad , \end{aligned} \tag{2.140}$$

   with parameters: ϵ, a, n, m, c.

4. Gupta potential [41]: (**gupt**) The Gupta potential is another analytical potential in the FSM class. It has the form:

$$\begin{aligned} V_{ij}(r_{ij}) &= A\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right) \\ \rho_{ij}(r_{ij}) &= \exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right) \\ F(\rho_i) &= -B\sqrt{\rho_i} \quad , \end{aligned} \tag{2.141}$$

   with parameters: A, r_0, p, B, q_{ij}.

All of these metal potentials can be decomposed into pair contributions and thus fit within the general tabulation scheme of DL_POLY Classic, where they are treated as pair interactions (though note that the metal cutoff, $r_{\rm met}$ has nothing to do with short ranged cutoff, $r_{\rm vdw}$). DL_POLY Classic calculates this potential in two stages: the first calculates the local density, ρ_i, for each atom; and the second calculates the potential energy and forces. Interpolation arrays, `vmet`, `gmet` and `fmet`

(METGEN, METTAB) are used in both these stages in the same spirit as in the van der Waals interaction calculations.

The total force $\underline{f}_k^{tot}$ on an atom k derived from this potential is calculated in the standard way:

$$\underline{f}_k^{tot} = -\underline{\nabla}_k U_{metal} \quad . \tag{2.142}$$

We rewrite the EAM/FSM potential, (2.137), as

$$\begin{aligned} U_{metal} &= U_1 + U_2 \\ U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} V_{ij}(r_{ij}) \\ U_2 &= \sum_{i=1}^{N} F(\rho_i) \quad , \end{aligned} \tag{2.143}$$

where $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. The force on atom k is the sum of the derivatives of U_1 and U_2 with respect to $\underline{r}_k$, which is recognisable as a sum of pair forces:

1. EAM force

$$\begin{aligned} -\frac{\partial U_1}{\partial \underline{r}_k} &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} \frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial \underline{r}_k} = \sum_{j=1,j\neq k}^{N} \frac{\partial V_{kj}(r_{kj})}{\partial r_{kj}}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \\ -\frac{\partial U_2}{\partial \underline{r}_k} &= -\sum_{i=1}^{N}\frac{\partial F}{\partial \rho_i}\sum_{j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial \underline{r}_k} \\ &= -\sum_{i=1,i\neq k}^{N}\frac{\partial F}{\partial \rho_i}\frac{\partial \rho_{ik}(r_{ik})}{\partial r_{ik}}\frac{\partial r_{ik}}{\partial \underline{r}_k} - \sum_{j=1,j\neq k}^{N}\frac{\partial F}{\partial \rho_k}\frac{\partial \rho_{kj}(r_{kj})}{\partial r_{kj}}\frac{\partial r_{kj}}{\partial \underline{r}_k} \\ &= \sum_{j=1,j\neq k}^{N}\left(\frac{\partial F}{\partial \rho_k} + \frac{\partial F}{\partial \rho_j}\right)\frac{\partial \rho_{kj}(r_{kj})}{\partial r_{kj}}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \quad . \end{aligned} \tag{2.144}$$

   In DL_POLY Classic the generation of the force arrays from tabulated data (implemented in the METAL_DERIV routine) is done using a five point interpolation precedure.

2. Finnis-Sinclair force

$$\begin{aligned} -\frac{\partial U_1}{\partial \underline{r}_k} &= \sum_{j=1,j\neq k}^{N}\left\{2(r_{kj}-c)(c_0+c_1 r_{kj}+c_2 r_{kj}^2) + (r_{kj}-c)^2(c_1+2c_2 r_{kj})\right\}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \\ -\frac{\partial U_2}{\partial \underline{r}_k} &= -\sum_{j=1,j\neq k}^{N}\frac{A}{2}\left(\frac{1}{\sqrt{\rho_k}}+\frac{1}{\sqrt{\rho_j}}\right)\left\{2(r_{kj}-d) + 3\beta\frac{(r_{kj}-d)^2}{d}\right\}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \quad . \end{aligned} \tag{2.145}$$

3. Sutton-Chen force

$$\begin{aligned} -\frac{\partial U_1}{\partial \underline{r}_k} &= -\sum_{j=1,j\neq k}^{N} n\epsilon\left(\frac{a}{r_{kj}}\right)^n \frac{\underline{\underline{r}}_{kj}}{r_{kj}^2} \\ -\frac{\partial U_2}{\partial \underline{r}_k} &= \sum_{j=1,j\neq k}^{N}\frac{mc\epsilon}{2}\left(\frac{1}{\sqrt{\rho_k}}+\frac{1}{\sqrt{\rho_j}}\right)\left(\frac{a}{r_{kj}}\right)^m \frac{\underline{\underline{r}}_{kj}}{r_{kj}^2} \quad . \end{aligned} \tag{2.146}$$

4. Gupta force

$$
\begin{aligned}
-\frac{\partial U_1}{\partial \underline{r}_k} &= -\sum_{j=1,j\neq k}^{N} \frac{Ap}{r_0} \exp\left(-p\frac{r_{kj}-r_0}{r_0}\right)\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \\
-\frac{\partial U_2}{\partial \underline{r}_k} &= \sum_{j=1,j\neq k}^{N} \frac{Bq_{kj}}{r_0}\left(\frac{1}{\sqrt{\rho_k}}+\frac{1}{\sqrt{\rho_j}}\right)\exp\left(-2q_{kj}\frac{r_{kj}-r_0}{r_0}\right)\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \quad . \qquad (2.147)
\end{aligned}
$$

With the metal forces thus defined the contribution to be added to the atomic virial *from each atom pair* is then

$$
\mathcal{W} = -\underline{r}_{ij}\cdot\underline{f}_j \quad , \qquad (2.148)
$$

which equates to:

$$
\begin{aligned}
\Psi &= 3V\frac{\partial U}{\partial V} \\
\Psi &= \frac{3}{2}V\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial V} + 3V\sum_{i=1}^{N}\frac{\partial F(\rho_i)}{\partial \rho_i}\frac{\partial \rho_i}{\partial V} = \Psi_1+\Psi_2 \\
&\frac{\partial r_{ij}}{\partial V} = \frac{\partial V^{1/3}s_{ij}}{\partial V} = \frac{1}{3}V^{-2/3}s_{ij} = \frac{r_{ij}}{3V} \\
\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \qquad (2.149) \\
&\frac{\partial \rho_i}{\partial V} = \frac{\partial}{\partial V}\sum_{j=1,j\neq i}^{N}\rho_{ij}(r_{ij}) = \sum_{j=1,j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial V} = \frac{1}{3V}\sum_{j=1,j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\
\Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\left(\frac{\partial F(\rho_i)}{\partial \rho_i}+\frac{\partial F(\rho_j)}{\partial \rho_j}\right)\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \quad .
\end{aligned}
$$

1. EAM virial
   The same as above.

2. Finnis-Sinclair virial

$$
\begin{aligned}
\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\left\{2(r_{ij}-c)(c_0+c_1r_{ij}+c_2r_{ij}^2)+(r_{ij}-c)^2(c_1+2c_2r_{ij})\right\}r_{ij} \\
\Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{A}{2}\left(\frac{1}{\sqrt{\rho_i}}+\frac{1}{\sqrt{\rho_j}}\right)\left\{2(r_{ij}-d)+3\beta\frac{(r_{ij}-d)^2}{d}\right\}r_{ij}a \quad . \qquad (2.150)
\end{aligned}
$$

3. Sutton-Chen virial

$$
\begin{aligned}
\Psi_1 &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}n\epsilon\left(\frac{a}{r_{ij}}\right)^n \\
\Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{mc\epsilon}{2}\left(\frac{1}{\sqrt{\rho_i}}+\frac{1}{\sqrt{\rho_j}}\right)\left(\frac{a}{r_{ij}}\right)^m \quad . \qquad (2.151)
\end{aligned}
$$

4. Gupta virial

$$\begin{aligned}\Psi_1 &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{Ap}{r_0}\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right)r_{ij} \\ \Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{Bq_{ij}}{r_0}\left(\frac{1}{\sqrt{\rho_i}}+\frac{1}{\sqrt{\rho_j}}\right)\exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right)r_{ij} \quad .\end{aligned} \tag{2.152}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta} \quad , \tag{2.153}$$

where α and β indicate the x, y, z components. The atomic stress tensor is symmetric.

The long ranged correction for the DL_POLY Classic metal potential is in two parts. Firstly, by analogy with the short ranged potentials, the correction to the local density is

$$\begin{aligned}\rho_i &= \sum_{j=1,j\neq i}^{\infty}\rho_{ij}(r_{ij}) \\ \rho_i &= \sum_{j=1,j\neq i}^{r_{ij}<r_{\rm met}}\rho_{ij}(r_{ij}) + \sum_{j=1,j\neq i}^{r_{ij}\geq r_{\rm met}}\rho_{ij}(r_{ij}) = \rho_i^o + \delta\rho_i \qquad (2.154) \\ \delta\rho_i &= 4\pi\bar{\rho}\int_{r_{\rm met}}^{\infty}\rho_{ij}(r)dr \quad , \qquad (2.155)\end{aligned}$$

where ρ_i^o is the uncorrected local density and $\bar{\rho}$ is the *mean particle density*. Evaluating the integral part of the above equation yields:

1. EAM density correction
   No long ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair density correction
   No long ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen density correction

$$\delta\rho_i = \frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{m-3} \quad . \tag{2.156}$$

4. Gupta density correction

$$\delta\rho_i = \frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^2 + 2r_{\rm met}\left(\frac{r_0}{q_{ij}}\right) + 2\left(\frac{r_0}{q_{ij}}\right)^2\right]\exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right) \quad . \tag{2.157}$$

The density correction is applied immediately after the local density is calculated. The pair term correction is obtained by analogy with the short ranged potentials and is

$$\begin{aligned}U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{\infty}V_{ij}(r_{ij}) \\ U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}<r_{\rm met}}V_{ij}(r_{ij}) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}}V_{ij}(r_{ij}) = U_1^o + \delta U_1\end{aligned}$$

$$
\begin{aligned}
\delta U_1 &= 2\pi N\bar{\rho}\int_{r_{\rm met}}^{\infty} V_{ij}(r)r^2dr \\
U_2 &= \sum_{i=1}^{N} F(\rho_i^0+\delta\rho_i) \\
U_2 &= \sum_{i=1}^{N} F(\rho_i^0) + \sum_{i=1}^{N}\frac{\partial F(\rho_i)_0}{\partial\rho_i}\delta\rho_i) = U_2^0+\delta U_2 \\
\delta U_2 &= 4\pi\bar{\rho}\sum_{i=1}^{N}\frac{\partial F(\rho_i)_0}{\partial\rho_i}\int_{r_{\rm met}}^{\infty}\rho_{ij}(r)r^2dr \quad .
\end{aligned}
\tag{2.158}
$$

Note: that $\delta U2$ is not required if ρ_i has already been corrected. Evaluating the integral part of the above equations yields:

1. EAM energy correction
   No long ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair energy correction
   No long ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen energy correction

$$
\begin{aligned}
\delta U_1 &= \frac{2\pi N\bar{\rho}\epsilon a^3}{(n-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3} \\
\delta U_2 &= -\frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3}\left\langle\frac{Nc\epsilon}{2\sqrt{\rho_i^0}}\right\rangle \quad .
\end{aligned}
\tag{2.159}
$$

4. Gupta energy correction

$$
\begin{aligned}
\delta U_1 &= \frac{2\pi N\bar{\rho}Ar_0}{p}\left[r_{\rm met}^2+2r_{\rm met}\left(\frac{r_0}{p}\right)+2\left(\frac{r_0}{p}\right)^2\right]\times \\
&\quad \exp\left(-p\frac{r_{\rm met}-r_0}{r_0}\right) \\
\delta U_2 &= -\frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^2+2r_{\rm met}\left(\frac{r_0}{q_{ij}}\right)+2\left(\frac{r_0}{q_{ij}}\right)^2\right]\times \\
&\quad \exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right)\left\langle\frac{NB}{2\sqrt{\rho_i^0}}\right\rangle \quad .
\end{aligned}
\tag{2.160}
$$

To estimate the virial correction we assume the corrected local densities are constants (i.e. independent of distance - at least beyond the ranged $r_{\rm met}$). This allows the virial correction to be computed by the methods used in the short ranged potentials:

$$
\begin{aligned}
\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{\infty}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\
\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}<r_{\rm met}}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij}+\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\
&= \Psi_1^0+\delta\Psi_1
\end{aligned}
$$

$$
\begin{aligned}
\delta\Psi_1 &= 2\pi N\bar{\rho}\int_{r_{\rm met}}^{\infty}\frac{\partial V_{ij}(r)}{\partial r_{ij}}r^3dr \\
\Psi_2 &= \sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{\infty}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\
\Psi_2 &= \sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{r_{ij}<r_{\rm met}}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij}+\sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\
&= \Psi_2^0+\delta\Psi_2 \\
\delta\Psi_2 &= 4\pi\bar{\rho}\sum_{i=1}^{N}\frac{\partial F(\rho_i)}{\partial\rho_i}\int_{r_{\rm met}}^{\infty}\frac{\partial\rho_{ij}(r)}{\partial r}r^3dr \quad .
\end{aligned}
\tag{2.161}
$$

Evaluating the integral part of the above equations yields:

1. EAM virial correction
   No long ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair virial correction
   No long ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen virial correction

$$
\begin{aligned}
\delta\Psi_1 &= -n\frac{2\pi N\bar{\rho}\epsilon a^3}{(n-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3} \\
\delta\Psi_2 &= m\frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3}\left\langle\frac{Nc\epsilon}{2\sqrt{\rho_i^0}}\right\rangle \quad .
\end{aligned}
\tag{2.162}
$$

4. Gupta virial correction

$$
\begin{aligned}
\delta\Psi_1 &= -\frac{p}{r_0}\frac{2\pi N\bar{\rho}Ar_0}{p}\left[r_{\rm met}^3+3r_{\rm met}^2\left(\frac{r_0}{p}\right)+6r_{\rm met}\left(\frac{r_0}{p}\right)^2+6\left(\frac{r_0}{p}\right)^3\right]\times \\
&\quad \exp\left(-p\frac{r_{\rm met}-r_0}{r_0}\right) \\
\delta\Psi_2 &= \frac{q_{ij}}{r_0}\frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^3+3r_{\rm met}^2\left(\frac{r_0}{q_{ij}}\right)+6r_{\rm met}\left(\frac{r_0}{q_{ij}}\right)^2+6\left(\frac{r_0}{q_{ij}}\right)^3\right]\times \\
&\quad \exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right)\left\langle\frac{NB}{2\sqrt{\rho_i^0}}\right\rangle \quad .
\end{aligned}
\tag{2.163}
$$

In the energy and virial corrections we have used the approximation:

$$
\sum_i^N\rho_i^{-1/2}=\frac{N}{<\rho_i^{1/2}>} \quad , \tag{2.164}
$$

where $<\rho_i^{1/2}>$ is regarded as a constant of the system.

In DL_POLY Classic the metal forces are handled by the routine METFRC. The local density is calculated by the routines METDENS, EAMDEN and FSDEN. The long ranged corrections are calculated by LRCMETAL. Reading and generation of EAM table data from TABEAM is handled by METTAB and METAL_DERIV.

Notes on the Treatment of Alloys

The distinction to be made between EAM and FSM potentials with regard to alloys concerns the mixing rules for unlike interactions. Starting with equations (2.137) and (2.138), it is clear that we require mixing rules for terms $V_{ij}(r_{ij})$ and $\rho_{ij}(r_{ij})$ when atoms i and j are of different kinds. Thus two different metals A and B we can distinguish 4 possible variants of each:

$$V_{ij}^{AA}(r_{ij}),\ V_{ij}^{BB}(r_{ij}),\ V_{ij}^{AB}(r_{ij}),\ V_{ij}^{BA}(r_{ij})$$

and

$$\rho_{ij}^{AA}(r_{ij}),\ \rho_{ij}^{BB}(r_{ij}),\ \rho_{ij}^{AB}(r_{ij}),\ \rho_{ij}^{BA}(r_{ij}).$$

These forms recognise that the contribution of a type A atom to the potential of a type B atom may be different from the contribution of a type B atom to the potential of a type A atom. In both EAM [4] and FSM [39] cases it turns out that

$$V_{ij}^{AB}(r_{ij}) = V_{ij}^{BA}(r_{ij}) \quad , \tag{2.165}$$

though the mixing rules are different in each case (**beware!**).

With regard to density, in the EAM case it is required that [4]:

$$\begin{aligned} \rho_{ij}^{AB}(r_{ij}) &= \rho_{ij}^{BB}(r_{ij}) \\ \rho_{ij}^{BA}(r_{ij}) &= \rho_{ij}^{AA}(r_{ij}) \quad , \end{aligned} \tag{2.166}$$

which means that an atom of type A contributes the same density to the environment of an atom of type B as it does to an atom of type A, and *vice versa*.

For the FSM case [39] a different rule applies:

$$\rho_{ij}^{AB}(r_{ij}) = (\rho_{ij}^{AA}(r_{ij})\rho_{ij}^{BB}(r_{ij}))^{1/2} \tag{2.167}$$

so that atoms of type A and B contribute the same densities to each other, but not to atoms of the same type.

Thus when specifying these potentials in the DL_POLY Classic FIELD file for an alloy composed of n different metal atom types both EAM and FSM require the specification of $n(n+1)/2$ pair functions $V_{ij}^{AB}(r_{ij})$. However, the EAM requires only n density functions $\rho_{ij}^{AA}(r_{ij})$, whereas the FSM class requires all the cross functions $\rho_{ij}^{AB}(r_{ij})$ or $n(n+1)/2$ in total. In addition to the $n(n+1)/2$ pair functions and n density functions the EAM requires further specification of n functional forms of the density dependence (i.e. the embedding function $F(\rho_i)$ in (2.137)).

For EAM potentials all the functions are supplied in tabular form via the table file TABEAM (see section 4.1.6) to which DL_POLY Classic is redirected by the FIELD file data. The FSM potentials are defined via the necessary parameters in the FIELD file.

2.3.6 External Fields

In addition to the molecular force field, DL_POLY Classic allows the use of an *external* force field. Examples of field available include:

1. Electric field: (**elec**)
$$\underline{F_i} = \underline{F_i} + q_i.\underline{H} \tag{2.168}$$
2. Oscillating shear: (**oshm**)
$$\underline{F}_x = A\cos(2n\pi.z/L_z) \tag{2.169}$$

3. Continuous shear: (**shrx**)
$$\underline{v}_x = \frac{1}{2} A \frac{|z|}{z} \qquad : |z| > z_0 \tag{2.170}$$
4. Gravitational field: (**grav**)
$$\underline{F_i} = \underline{F_i} + m_i.\underline{H} \tag{2.171}$$
5. Magnetic field: (**magn**)
$$\underline{F_i} = \underline{F_i} + q_i.(\underline{v_i} \wedge \underline{H}) \tag{2.172}$$
6. Containing sphere: (**sphr**)
$$\underline{F} = A(R_0 - r)^{-n} \qquad : r > R_{cut} \tag{2.173}$$
7. Harmonic repulsive wall in z-direction: (**zbnd**)
$$\underline{F} = A(z_o - z) \qquad : z > z_o \tag{2.174}$$
8. Harmonic restraint zone in z-direction: (**zres**)
$$\underline{F} = A(z_{max} - z_{com}) \qquad : z_{com} > z_{max} \tag{2.175}$$
$$\underline{F} = A(z_{min} - z_{com}) \qquad : z_{com} < z_{min} \tag{2.176}$$
where z_{com} is the chosen molecule centre of mass.

It is recommended that the use of an external field should be accompanied by a thermostat (this does not apply to examples 6 and 7, since these are conservative fields). The user is advised to be careful with units!

In DL_POLY Classic external field forces are handled by the routine EXTNFLD.

2.4 Long Ranged Electrostatic (Coulombic) Potentials

DL_POLY Classic incorporates several techniques for dealing with long ranged electrostatic potentials [2]. These are as follows.

1. Atomistic and charge group implementation.
2. Direct Coulomb sum;
3. Truncated and shifted Coulomb sum;
4. Damped shifted force Coulomb sum;
5. Coulomb sum with distance dependent dielectric;
6. Ewald sum;
7. Smoothed Particle Mesh Ewald (SPME);
8. Hautman Klein Ewald for systems with 2D periodicity;
9. Reaction field;

[2] Unlike the other elements of the force field, the electrostatic forces are NOT specified in the input FIELD file, but by setting appropriate directives in the CONTROL file. See section 4.1.1.

10. Dynamical shell model;

11. Relaxed shell model.

Some of these techniques can be combined. For example 1, 3 and 4 can be used in conjunction with 9. The Ewald sum, SPME and Hautman Klein Ewald are restricted to periodic (or pseudo-periodic) systems only, though DL_POLY Classic can handle a broad selection of periodic boundary conditions, including cubic, orthorhombic, parallelepiped, truncated octahedral, hexagonal prism and rhombic dodecahedral. The Ewald sum is the method of choice for periodic systems. The other techniques can be used with either periodic or non-periodic systems, though in the case of the direct Coulomb sum, there are likely to be problems with convergence.

DL_POLY Classic will correctly handle the electrostatics of both molecular and atomic species. However it is assumed that the system is electrically neutral. A warning message is printed if the system is found to be charged, but otherwise the simulation proceeds as normal. No correction for non-neutrality is applied, except in the case of the Ewald based methods.

2.4.1 Atomistic and Charge Group Implementation

The Ewald sum is an accurate method for summing long ranged Coulomb potentials in periodic systems. This can be a very cpu intensive calculation and the use of more efficient, but less accurate methods, is common. Invariably this involves truncation of the potential at some finite distance $r_{\rm cut}$. If an atomistic scheme is used for the truncation criterion there is no guarantee that the interaction sphere will be neutral and spurious "charging" effects will almost certainly be seen in a simulation. This arises because the potential being truncated is long ranged ($1/r$ for charge-charge interactions). However if the cutoff scheme is based on *neutral* groups of atoms, then at worst, at long distance the interaction will be a dipole-dipole interaction and vary as $1/r^3$. The truncation effects at the cutoff are therefore much less severe than if an atomistic scheme is used. In DL_POLY Classic the interaction is evaluated between all atoms of both groups if any site of the first group is within the cutoff distance of any site of the second group. The groups are known interchangeably as "charge groups" or "neutral groups" in the documentation - which serves as a reminder that the advantages of using such a scheme are lost if the groups carry an overall charge. There is no formal requirement in DL_POLY Classic that the groups actually be electrically neutral.

The charge group scheme is more cpu intensive than a simple atomistic cutoff scheme as more computation is required to determine whether or not to include a set of interactions. However the size of the Verlet neighbourhood list (easily the largest array in DL_POLY Classic) is considerably smaller with a charge group scheme than an atomistic scheme as only a list of interacting groups need be stored as opposed to a list of interacting atoms.

2.4.2 Direct Coulomb Sum

Use of the direct Coulomb sum is sometimes necessary for accurate simulation of isolated (nonperiodic) systems. It is *not* recommended for periodic systems.

The interaction potential for two charged ions is

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{2.177}$$

with q_ℓ the charge on an atom labelled ℓ, and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$.

The force on an atom j derived from this force is

$$\underline{f}_j = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}^3}\underline{r}_{ij} \tag{2.178}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{2.179}$$

which is simply the negative of the potential term.

The contribution to be added to the atomic stress tensor is

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.180}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY Classic these forces are handled by the routines COUL0 and COUL0NEU.

2.4.3 Truncated and Shifted Coulomb Sum

This form of the Coulomb sum has the advantage that it drastically reduces the ranged of electrostatic interactions, without giving rise to a violent step in the potential energy at the cutoff. Its main use is for preliminary preparation of systems and it is not recommended for realistic models.

The form of the potential function is

$$U(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0}\left\{\frac{1}{r_{ij}} - \frac{1}{r_{cut}}\right\} \tag{2.181}$$

with q_ℓ the charge on an atom labelled ℓ, r_{cut} the cutoff radius and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$.

The force on an atom j derived from this potential, within the radius r_{cut}, is

$$\underline{f}_j = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}^3}\underline{r}_{ij} \tag{2.182}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j \tag{2.183}$$

which is *not* the negative of the potential term in this case.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.184}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY Classic these forces are handled by the routine COUL1.

2.4.4 Damped Shifted Force Coulomb sum

A further refinement of the truncated and shifted Coulomb sum is to truncate the $1/r$ potential at $r_{\rm cut}$ and add a linear term to the potential in order to make both the energy and the force zero at the cutoff (the shifted force Coulombic potential). This is formally equivalent to surrounding each charge with a spherical charge of radius r_{cut}, which neutralises the charge content of the cutoff sphere. The potential is thus

$$U(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{1}{r_{ij}} + \frac{r_{ij}}{r_{\rm cut}^2} - \frac{2}{r_{\rm cut}}\right] \tag{2.185}$$

with the force on atom j given by

$$\underline{f}_j = \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{1}{r_{ij}^2} - \frac{1}{r_{\rm cut}^2}\right]\frac{\underline{r}_{ij}}{r_{ij}} \tag{2.186}$$

with the force on atom i the negative of this.

This removes the heating effects that arise from the discontinuity in the forces at the cutoff in the simple truncated and shifted potential.

More recently Wolf *et al* [42] took the shifted force Coulomb potential a step further by the introduction of an additional 'damping' function to moderate the $1/r_{ij}$ dependence . This was reported to be a viable alternative to the Ewald summation that was particularly effective for large systems. The basic assumption is that in condensed phase systems the electrostatic forces are effectively screened by charge ordering so that at long ranged any given charge 'looks' like a neutral object. Meanwhile the force shifting is formally equivalent to surrounding each charge with a spherical charge that neutralises the charge content of the cutoff sphere, thus resembling the natural screening on a predetermined distance scale (r_{cut}). The method thus assumes that these two effects are the same.

The Wolf *et al* method [42] was cast into a form suitable for molecular dynamics by Fennell and Gezelter [43], which is the form implemented in DL_POLY Classic. In this form damping function is the same complementary error function as appears in the Ewald sum (see section 2.4.6):

$$\begin{aligned} U(r_{ij}) \;=\; & \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{erfc(\alpha r_{ij})}{r_{ij}} - \frac{erfc(\alpha r_{cut})}{r_{cut}} + \left(\frac{erfc(\alpha r_{cut})}{r_{cut}^2} + \frac{2\alpha}{\pi^{1/2}}\frac{exp(-\alpha^2 r_{cut}^2)}{r_{cut}}\right)(r_{ij} - r_{cut})\right]\cdots \\ & \cdots\cdots\cdots\

The interatomic potential for two charged ions is

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0\epsilon(r_{ij})}\frac{q_i q_j}{r_{ij}} \tag{2.191}$$

with q_ℓ the charge on an atom labelled ℓ, and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. $\epsilon(r)$ is the distance dependent dielectric function. In DL_POLY Classic it is assumed that this function has the form

$$\epsilon(r) = \epsilon r \tag{2.192}$$

where ϵ is a constant. Inclusion of this term effectively accelerates the rate of convergence of the Coulomb sum.

The force on an atom j derived from this potential is

$$\underline{f}_j = \frac{1}{2\pi\epsilon_0\epsilon}\frac{q_i q_j}{r_{ij}^4}\underline{r}_{ij} \tag{2.193}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\underline{r}_{ij}\cdot\underline{f}_j \tag{2.194}$$

which is -2 times the potential term.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.195}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY Classic these forces are handled by the routines COUL2 and COUL2NEU.

One last point to note is that the reaction field method can also be implemented with the damped shifted force Coulombic potential described above (section 2.4.4), so that polarisation of the long ranged medium by the dipole of the cutoff sphere may be accounted for.

2.4.6 Ewald Sum

The Ewald sum [12] is the best technique for calculating electrostatic interactions in a periodic (or pseudo-periodic) system.

The basic model for a neutral periodic system is a system of charged point ions mutually interacting via the Coulomb potential. The Ewald method makes two amendments to this simple model. Firstly each ion is effectively neutralised (at long range) by the superposition of a spherical gaussian cloud of opposite charge centred on the ion. The combined assembly of point ions and gaussian charges becomes the *Real Space* part of the Ewald sum, which is now short ranged and treatable by the methods described above (section 2.1). [3] The second modification is to superimpose a second set of gaussian charges, this time with the same charges as the original point ions and again centred on the point ions (so nullifying the effect of the first set of gaussians). The potential due to these gaussians is obtained from Poisson's equation and is solved as a Fourier series in *Reciprocal Space*. The complete Ewald sum requires an additional correction, known as the self energy correction, which arises from a gaussian acting on its own site, and is constant. Ewald's method therefore replaces a potentially infinite sum in real space by two finite sums: one in real space and one in reciprocal space; and the self energy correction.

[3] Strictly speaking, the real space sum ranges over all periodic images of the simulation cell, but in the DL_POLY Classic implementation, the parameters are chosen to restrict the sum to the simulation cell and its nearest neighbours i.e. the *minimum images* of the cell contents.

For molecular systems, as opposed to systems comprised simply of point ions, additional modifications are necessary to correct for the excluded (intra-molecular) Coulombic interactions. In the real space sum these are simply omitted. In reciprocal space however, the effects of individual gaussian charges cannot easily be extracted, and the correction is made in real space. It amounts to removing terms corresponding to the potential energy of an ion ℓ due to the gaussian charge on a neighbouring charge m (or *vice versa*). This correction appears as the final term in the full Ewald formula below. The distinction between the error function erf and the more usual complementary error function $erfc$ found in the real space sum, should be noted.

The total electrostatic energy is given by the following formula.

$$\begin{aligned} U_c &= \frac{1}{2V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}\frac{\exp(-k^2/4\alpha^2)}{k^2}|\sum_j^N q_j\exp(-i\underline{k}\cdot\underline{r}_j)|^2+\frac{1}{4\pi\epsilon_0}\sum_{n<j}^{N^*}\frac{q_jq_n}{r_{nj}}erfc(\alpha r_{nj}) \\ &-\frac{1}{4\pi\epsilon_0}\sum_{molecules}\sum_{\ell\leq m}^{M^*}q_\ell q_m\left\{\delta_{\ell m}\frac{\alpha}{\sqrt{\pi}}+\frac{erf(\alpha r_{\ell m})}{r_{\ell m}^{1-\delta_{\ell m}}}\right\}-\frac{1}{4\pi\epsilon}\frac{1}{V_o\alpha^2}\left\{\sum_j^N q_j\right\}^2, \end{aligned} \tag{2.196}$$

where N is the number of ions in the system and N^* the same number discounting any excluded (intramolecular) interactions. M^* represents the number of excluded atoms in a given molecule and includes the atomic self correction. The final term on the right is the Fuchs correction for charged systems [44]. V_o is the simulation cell volume and $\underline{k}$ is a reciprocal lattice vector defined by

$$\underline{k}=\ell\underline{u}+m\underline{v}+n\underline{w} \tag{2.197}$$

where ℓ, m, n are integers and $\underline{u},\underline{v},\underline{w}$ are the *reciprocal space* basis vectors. Both V_o and $\underline{u},\underline{v},\underline{w}$ are derived from the vectors $(\underline{a},\underline{b},\underline{c})$ defining the simulation cell. Thus

$$V_o=|\underline{a}\cdot\underline{b}\times\underline{c}| \tag{2.198}$$

and

$$\begin{aligned} \underline{u} &= 2\pi\frac{\underline{b}\times\underline{c}}{\underline{a}\cdot\underline{b}\times\underline{c}} \\ \underline{v} &= 2\pi\frac{\underline{c}\times\underline{a}}{\underline{a}\cdot\underline{b}\times\underline{c}} \\ \underline{w} &= 2\pi\frac{\underline{a}\times\underline{b}}{\underline{a}\cdot\underline{b}\times\underline{c}}. \end{aligned} \tag{2.199}$$

With these definitions, the Ewald formula above is applicable to general periodic systems. (A small additional modification is necessary for rhombic dodecahedral and truncated octahedral simulation cells [45].)

In practice the convergence of the Ewald sum is controlled by three variables: the real space cutoff r_{cut}; the convergence parameter α and the largest reciprocal space vector $\underline{k}_{max}$ used in the reciprocal space sum. These are discussed more fully in section 3.2.5. DL_POLY Classic can provide estimates if requested (see CONTROL file description 4.1.1.

The force on an atom j is obtained by differentiation and is

$$\begin{aligned} \underline{f}_j &= -\frac{q_j}{V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}i\underline{k}\exp(i\underline{k}\cdot\underline{r}_j)\frac{\exp(-k^2/4\alpha^2)}{k^2}\sum_n^N q_n\exp(-i\underline{k}\cdot\underline{r}_n) \\ &+\frac{q_j}{4\pi\epsilon_0}\sum_n^{N^*}\frac{q_n}{r_{nj}^3}\left\{erfc(\alpha r_{nj})+\frac{2\alpha r_{nj}}{\sqrt{\pi}}\exp(-\alpha^2r_{nj}^2)\right\}\underline{r}_{nj} \\ &-\frac{q_j}{4\pi\epsilon_0}\sum_\ell^{M^*}\frac{q_\ell}{r_{\ell j}^3}\left\{erf(\alpha r_{\ell j})-\frac{2\alpha r_{\ell j}}{\sqrt{\pi}}\exp(-\alpha^2r_{\ell j}^2)\right\}\underline{r}_{\ell j} \end{aligned} \tag{2.200}$$

The electrostatic contribution to the system virial can be obtained as the negative of the Coulombic energy. However in DL_POLY Classic this formal equality can be used as a check on the convergence of the Ewald sum. The actual electrostatic virial is obtained during the calculation of the diagonal of the stress tensor.

The electrostatic contribution to the stress tensor is given by

$$\begin{aligned}\underline{\underline{\sigma}} &= \frac{1}{2V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}\left\{\underline{\underline{\mathbf{1}}}-2\left(\frac{1}{4\alpha^2}+\frac{1}{k^2}\right)\underline{\underline{\mathbf{K}}}\right\}\frac{\exp(-k^2/4\alpha^2)}{k^2}|\sum_{j}^{N}q_j\exp(-i\underline{k}\cdot\underline{r}_j)|^2 \\ &+\frac{1}{4\pi\epsilon_0}\sum_{j<n}^{N^*}\frac{q_jq_n}{r_{nj}^3}\left\{erfc(\alpha r_{nj})+\frac{2\alpha r_{nj}}{\sqrt{\pi}}\exp(-\alpha^2r_{nj}^2)\right\}\underline{\underline{\mathbf{R_{nj}}}} \\ &-\frac{1}{4\pi\epsilon_0}\sum_{j<\ell}^{M^*}\frac{q_jq_\ell}{r_{\ell j}^3}\left\{erf(\alpha r_{\ell j})-\frac{2\alpha r_{\ell j}}{\sqrt{\pi}}\exp(-\alpha^2r_{\ell j}^2)\right\}\underline{\underline{\mathbf{R_{\ell j}}}},\end{aligned} \tag{2.201}$$

where matrices $\underline{\underline{\mathbf{K}}}$ and $\underline{\underline{\mathbf{R_{\ell j}}}}$ are defined as follows.

$$K^{\alpha\beta} = k^\alpha k^\beta \tag{2.202}$$

$$R_{\ell j}^{\alpha\beta} = r_{\ell j}^\alpha r_{\ell j}^\beta \tag{2.203}$$

In DL_POLY Classic the full Ewald sum is handled by several routines: EWALD1 and EWALD1A handle the reciprocal space terms; EWALD2, EWALD2_2PT, EWALD2_RSQ and EWALD4, EWALD4_2PT handle the real space terms (with the same Verlet neighbour list routines that are used to calculate the short ranged forces); and EWALD3 calculates the self interaction corrections. It should be noted that the Ewald potential and force interpolation arrays in DL_POLY Classic are `erc` and `fer` respectively.

2.4.7 Smoothed Particle Mesh Ewald

As its name implies the Smoothed Particle Mesh Ewald (SPME) method is a modification of the standard Ewald method. DL_POLY Classic implements the SPME method of Essmann *et al.* [46]. Formally this method is capable of treating van der Waals forces also, but in DL_POLY Classic it is confined to electrostatic forces only. The main difference from the standard Ewald method is in its treatment of the the reciprocal space terms. By means of an interpolation procedure involving (complex) B-splines, the sum in reciprocal space is represented on a three dimensional rectangular grid. In this form the Fast Fourier Transform (FFT) may be used to perform the primary mathematical operation, which is a 3D convolution. The efficiency of these procedures greatly reduces the cost of the reciprocal space sum when the range of $\underline{k}$ vectors is large. The method (briefly) is as follows (for full details see [46]):

1. Interpolation of the $exp(-i\underline{k}\cdot\underline{r}_j)$ terms (given here for one dimension):

$$exp(2\pi iu_jk/L)\approx b(k)\sum_{\ell=-\infty}^{\infty}M_n(u_j-\ell)exp(2\pi ik\ell/K) \tag{2.204}$$

in which k is the integer index of the $\underline{k}$ vector in a principal direction, K is the total number of grid points in the same direction and u_j is the fractional coordinate of ion j scaled by a factor K (i.e. $u_j = Ks_j^x$). Note that the definition of the B-splines implies a dependence

on the integer K, which limits the formally infinite sum over ℓ. The coefficients $M_n(u)$ are B-splines of order n and the factor $b(k)$ is a constant computable from the formula:

$$b(k) = exp(2\pi i(n-1)k/K) \left[\sum_{\ell=0}^{n-2} M_n(\ell+1) exp(2\pi i k\ell/K) \right]^{-1} \tag{2.205}$$

2. Approximation of the structure factor $S(\underline{k})$:

$$S(\underline{k}) \approx b_1(k_1) b_2(k_2) b_3(k_3) Q^\dagger(k_1, k_2, k_3) \tag{2.206}$$

where $Q^\dagger(k_1, k_2, k_3)$ is the discrete Fourier transform of the *charge array* $Q(\ell_1, \ell_2, \ell_3)$ defined as

$$Q(\ell_1, \ell_2, \ell_3) = \sum_{j=1}^{N} q_j \sum_{n_1, n_2, n_3} M_n(u_{1j} - \ell_1 - n_1 L_1) M_n(u_{2j} - \ell_2 - n_2 L_2) M_n(u_{3j} - \ell_3 - n_3 L_3) \tag{2.207}$$

(in which the sums over $n_{1,2,3}$ etc are required to capture contributions from all relevant periodic cell images, which in practice means the nearest images.)

3. Approximating the reciprocal space energy U_{recip}:

$$U_{recip} = \frac{1}{2V_o\epsilon_0} \sum_{k_1, k_2, k_3} G^\dagger(k_1, k_2, k_3) Q(k_1, k_2, k_3) \tag{2.208}$$

in which $G^\dagger$ is the discrete Fourier transform of the function

$$G(k_1, k_2, k_3) = \frac{\exp(-k^2/4\alpha^2)}{k^2} B(k_1, k_2, k_3) (Q^\dagger(k_1, k_2, k_3))^* \tag{2.209}$$

and where

$$B(k_1, k_2, k_3) = |b_1(k_1)|^2 |b_2(k_2)|^2 |b_3(k_3)|^2 \tag{2.210}$$

and $(Q^\dagger(k_1, k_2, k_3))^*$ is the complex conjgate of $Q^\dagger(k_1, k_2, k_3)$. The function $G(k_1, k_2, k_3)$ is thus a relatively simple product of the gaussian screening term appearing in the conventional Ewald sum, the function $B(k_1, k_2, k_3)$ and the discrete Fourier transform of $Q(k_1, k_2, k_3)$

4. Calculating the atomic forces, which are given formally by:

$$f_j^\alpha = -\frac{\partial U_{recip}}{\partial r_j^\alpha} = -\frac{1}{V_o\epsilon_0} \sum_{k_1, k_2, k_3} G^\dagger(k_1, k_2, k_3) \frac{\partial Q(k_1, k_2, k_3)}{\partial r_j^\alpha} \tag{2.211}$$

Fortunately, due to the recursive properties of the B-splines, these formulae are easily evaluated.

The virial and the stress tensor are calculated in the same manner as for the conventional Ewald sum.

The DL_POLY Classic subroutines required to calculate the SPME contributions are: BSPGEN, which calculates the B-splines; BSPCOE, which calculates B-spline coefficients; SPL_CEXP, which calculates the FFT and B-spline complex exponentials; EWALD_SPME, which calculates the reciprocal space contributions; SPME_FOR, which calculates the reciprocal space forces; and DLPFFT3, which calculates the 3D complex fast Fourier transform (default code only, Cray, SGI, IBM SP machines have their own FFT routines, selected at compile time and the FFTW public FFT is also an option). These subroutines calculate the reciprocal space components of the Ewald sum only, the real-space calculations are performed by EWALD2, EWALD3 and EWALD 4, as for the normal Ewald sum. In addition there are a few minor utility routines : CPY_RTC copies a real array to a complex array; ELE_PRD is an element-for-element product of two arrays; SCL_CSUM is a scalar sum of elements of a complex array; and SET_BLOCK initialises an array to a preset value (usually zero).

2.4.8 Hautman Klein Ewald (HKE)

The method of Hautman and Klein is an adaptation of the Ewald method for systems which are periodic in two dimensions only [47]. (DL_POLY Classic assumes this periodicity is in the XY plane.)

The HKE method gives the following formula for the electrostatic energy of a system of N (nonbonded) ions that is overall charge neutral[4]:

$$\begin{aligned} U_c &= \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{i,j}^{N} q_i q_j z_{ij}^{2n} \sum_{\underline{g}\neq\underline{0}} f_n(g;\alpha) g^{2n-1} exp(i\underline{g}\cdot s_{ij}) + \\ &\frac{1}{8\pi\epsilon_0}\sum_{i\neq j}^{N} q_i q_j \sum_{\underline{L}} \left(\frac{1}{r_{ij,L}} - \sum_{n}^{n_{max}} a_n z_{ij}^{2n} \frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}} \right) + \\ &\frac{1}{8\pi\epsilon_0}\sum_{i}^{N} q_i^2 \sum_{\underline{L}} \frac{(1-h_0(L;\alpha))}{L} - \frac{\alpha}{\epsilon_0 \pi^{3/2}} \sum_{i}^{N} q_i^2 \end{aligned} \tag{2.212}$$

In this formula A is the system area (in the XY plane), $\underline{L}$ is a 2D lattice vector representing the 2D periodicity of the system, s_{ij} is the in-plane (XY) component of the interparticle distance r_{ij} and $\underline{g}$ is a reciprocal lattice vector. Thus

$$\underline{L} = \ell_1 \underline{a} + \ell_2 \underline{b}, \tag{2.213}$$

where ℓ_1, ℓ_2 are integers and vectors $\underline{a}$ and $\underline{b}$ are the lattice basis vectors. The reciprocal lattice vectors are:

$$\underline{g} = n_1 \underline{u} + n_2 \underline{v} \tag{2.214}$$

where n_1, n_2 are integers $\underline{u}, \underline{v}$ are reciprocal space vectors (defined in terms of the vectors $\underline{a}$ and $\underline{b}$):

$$\begin{aligned} \underline{u} &= 2\pi(b_y, -b_x)^\dagger / (a_x b_y - a_y b_x) \\ \underline{v} &= 2\pi(-a_y, a_x)^\dagger / (a_x b_y - a_y b_x). \end{aligned} \tag{2.215}$$

The functions $h_n(s;\alpha)$ and $f_n(s;\alpha)$ are the HKE convergence functions, in real and reciprocal space respectively. (C.f. the complementary error and gaussian functions of the original Ewald method.) However they occur to higher orders here, as indicated by the sum over subscript n, which corresponds to terms in a Taylor expansion of r^{-1} in s, the in-plane distance [47]. Usually this sum is truncated at $n_{max} = 1$, but in DL_POLY Classic can go as high as $n_{max} = 3$. In the HKE method the convergence functions are defined as follows:

$$h_n(s;\alpha)/s^{2n+1} = \frac{1}{a_n (2n)!} \nabla^{2n} (h_0(s;\alpha)/s) \tag{2.216}$$

with

$$h_0(s;\alpha) = erf(\alpha s) \tag{2.217}$$

and

$$f_n(g;\alpha) = \frac{1}{a_n (2n)!} f_0(g;\alpha) \tag{2.218}$$

with

$$f_0(g;\alpha) = erfc(g/2\alpha). \tag{2.219}$$

[4]The reader is warned that for the purpose of compatibility with other DL_POLY Classic Ewald routines we have defined $\alpha = 0.5/\alpha_{HK}$, where α_{HK} is the α parameter defined by Hautman and Klein in [47].

In DL_POLY Classic the $h_n(s;\alpha)/s^{2n+1}$ functions are derived by a recursion algorithm, while the $f_n(g;\alpha)$ functions are obtained by direct evaluation. The coefficients a_n are given by

$$a_n = (-1)^n(2n)!/(2^{2n}(n!)^2). \tag{2.220}$$

As pointed out by Hautman and Klein, the equation (2.212) allows separation of the z_{ij}^{2n} components via the binomial expansion, which greatly simplifies the double sum over atoms in reciprocal space. Thus the reciprocal space part of equation (2.212) becomes

$$U_{recip} = \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{\underline{g}\neq\underline{0}} f_n(g;\alpha) g^{2n-1} \sum_{p=0}^{2n}(-1)^p C_p^{2n} Z_p(\underline{g}) Z_{2n-p}^*(\underline{g}) \tag{2.221}$$

with C_p^{2n} a binomial coefficient and

$$Z_p(\underline{g}) = \sum_{j=1}^{N} q_j z_j^p exp(i\underline{g}\cdot\underline{s_j}) \tag{2.222}$$

The force on an ion is obtained by the usual differentiation, however in this case the z components have different expressions from the x and y.

$$\begin{aligned} -\frac{\partial U_c}{\partial u_j} &= \frac{1}{4\epsilon_0 A}\sum_{\underline{g}\neq\underline{0}}\sum_{n=0}^{n_{max}} a_n f_n(g;\alpha) g^{2n-1} \sum_{p=0}^{2n}(-1)^p C_p^{2n}\left(Z_p(\underline{g})\frac{\partial Z_{2n-p}^*(\underline{g})}{\partial u_j} + Z_{2n-p}^*(\underline{g})\frac{\partial Z_p(\underline{g})}{\partial u_j}\right) \\ &+\frac{q_j}{4\pi\epsilon_0}\sum_{n=0}^{n_{max}}\sum_{\underline{L}}\sum_{ij}{}' a_n q_i \frac{\partial}{\partial u_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) \end{aligned} \tag{2.223}$$

where u_j is one of x_j, y_j, z_j and (noting for brevity that x and y derivatives are similar)

$$\begin{aligned} \frac{\partial Z_p(\underline{g})}{\partial x_j} &= ig_x q_j z_j^p exp(i\underline{g}\cdot\underline{s_j}) \\ \frac{\partial Z_p(\underline{g})}{\partial z_j} &= pq_j z_j^{p-1} exp(i\underline{g}\cdot\underline{s_j}) \end{aligned} \tag{2.224}$$

and

$$\begin{aligned} \frac{\partial}{\partial x_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) &= s_{ij,L}^x \frac{z_{ij,L}^{2n}}{s_{ij,L}}\frac{\partial}{\partial x_j}\left(\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) \\ \frac{\partial}{\partial z_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) &= 2n z_{ij,L}^{2n-1}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}. \end{aligned} \tag{2.225}$$

In DL_POLY Classic the partial derivatives of $h_n(s_{ij,L};\alpha)/s_{ij,L}^{2n+1}$ are calculated by a recursion algorithm. Note that when $n=0$ there is no derivative w.r.t. z.

The virial and stress tensor terms in real space may be calculated directly from the pair forces and interatomic distances in the usual way, and need not be discussed further. The calculation of the reciprocal space contributions (the terms involving the $f_n(g;\alpha)$ functions) are more difficult. Firstly however we note that the reciprocal space contributions to σ_{xz}, σ_{yz} and σ_{zz} may be obtained directly from the force calculations thus:

$$\begin{aligned} \sigma_{xz}^{recip} &= \sum_j z_j f_j^x \\ \sigma_{yz}^{recip} &= \sum_j z_j f_j^y \\ \sigma_{zz}^{recip} &= \sum_j z_j f_j^z \end{aligned} \tag{2.226}$$

which renders the calculation of these components trivial. The remaining components are calculated from

$$\begin{aligned}\sigma_{uv}^{recip} &= U_{recip}\delta_{uv} + \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{\underline{g}\neq\underline{0}} g_u g_v \frac{g^{2n-2}}{a_n(2n)!} \\ &\left(\frac{(2n-1)f_0(g;\alpha)}{g} - \frac{1}{\alpha\sqrt{\pi}} exp(-g^2/4\alpha^2)\right) \\ &\sum_{p=0}^{2n}(-1)^p C_p^{2n} Z_p(\underline{g}) Z_{2n-p}^*(\underline{g})\end{aligned} \tag{2.227}$$

where u, v are one or both of the components x, y. Note that, although it is possible to define these contributions to the stress tensor, it is not possible to calculate a pressure from them unless a finite, arbitrary boundary is imposed on the z direction (which is an assumption applied in DL_POLY Classic, but without implications of periodicity in the z-direction). The x, y components define the surface tension however.

For bonded molecules, as with the standard 3D Ewald sum, it is necessary to extract contributions associated with the excluded atom pairs. In the DL_POLY Classic HKE implementation this amounts to an *a posteriori* subtraction of the corresponding coulomb terms.

In DL_POLY Classic the HKE method is handled by several subroutines: HKGEN constructs the $h_n(s;\alpha)$ convergence functions and their derivatives; HKEWALD1 calculates the reciprocal space terms; HKEWALD2 and HKEWALD3 calculate the real space terms and the bonded atom corrections respectively. HKEWALD4 calculates the primary interactions in the multiple timestep implementation.

2.4.9 Reaction Field

In the reaction field method it is assumed that any given molecule is surrounded by a spherical cavity of finite radius within which the electrostatic interactions are calculated explicitly. Outside the cavity the system is treated as a dielectric continuum. The occurence of any net dipole within the cavity induces a polarisation in the dielectric, which in turn interacts with the given molecule. The model allows the replacement of the infinite Coulomb sum by a finite sum plus the reaction field.

The reaction field model coded into DL_POLY Classic is the implementation of Neumann based on charge-charge interactions [48]. In this model, the total Coulombic potential is given by

$$U_c = \frac{1}{4\pi\epsilon_0}\sum_{j<n} q_j q_n \left[\frac{1}{r_{nj}} + \frac{B_0 r_{nj}^2}{2R_c^3}\right] \tag{2.228}$$

where the second term on the right is the reaction field correction to the explicit sum, with R_c the radius of the cavity. The constant B_0 is defined as

$$B_0 = \frac{2(\epsilon_1 - 1)}{(2\epsilon_1 + 1)}, \tag{2.229}$$

with ϵ_1 the dielectric constant outside the cavity. The effective pair potential is therefore

$$U(r_{nj}) = \frac{1}{4\pi\epsilon_0} q_j q_n \left[\frac{1}{r_{nj}} + \frac{B_0 r_{nj}^2}{2R_c^3}\right]. \tag{2.230}$$

This expression unfortunately leads to large fluctuations in the system Coulombic energy, due to the large 'step' in the function at the cavity boundary. In DL_POLY Classic this is countered by

subtracting the value of the potential at the cavity boundary from each pair contribution. The term subtracted is

$$\frac{1}{4\pi\epsilon_0}\frac{q_j q_n}{R_c}\left[1+\frac{B_0}{2}\right]. \tag{2.231}$$

The effective pair force on an atom j arising from another atom n within the cavity is given by

$$\underline{f}_j = \frac{q_j q_n}{4\pi\epsilon_0}\left[\frac{1}{r_{nj}^3}-\frac{B_0}{R_c^3}\right]\underline{r}_{nj}. \tag{2.232}$$

The contribution of each effective pair interaction to the atomic virial is

$$\mathcal{W} = -\underline{r}_{nj}\cdot\underline{f}_j \tag{2.233}$$

and the contribution to the atomic stress tensor is

$$\sigma^{\alpha\beta} = r_{nj}^{\alpha} f_j^{\beta}. \tag{2.234}$$

In DL_POLY Classic the reaction field is handled by the routines COUL3 and COUL3NEU.

2.4.10 Dynamical Shell Model

An atom or ion is polarisable if it develops a dipole moment when placed in an electric field. It is commonly expressed by the equation

$$\underline{\mu} = \alpha\underline{E}, \tag{2.235}$$

where $\underline{\mu}$ is the induced dipole and $\underline{E}$ is the electric field. The constant α is the polarisability.

The dynamical shell model is a method of incorporating polarisability into a molecular dynamics simulation. The method used in DL_POLY Classic is that devised by Fincham *et al* [49] and is known as the adiabatic shell model.

In the *static* shell model a polarisable atom is represented by a massive core and massless shell, connected by a harmonic spring, hereafter called the core-shell unit. The core and shell carry different electric charges, the sum of which equals the charge on the original atom. There is no electrostatic interaction (i.e. self interaction) between the core and shell of the same atom. Non-Coulombic interactions arise from the shell alone.

The harmonic spring has a potential of the form

$$V_{spring}(r_{ij}) = \frac{1}{2}kr_{ij}^2 \tag{2.236}$$

Sometimes an anharmonic spring is used, which is quartic in form:

$$V_{spring}(r_{ij}) = \frac{1}{2}kr_{ij}^2 + \frac{1}{4}k_4 r_{ij}^4. \tag{2.237}$$

Normally k is much larger than k_4.

The effect of an electric field is to separate the core and shell, giving rise to a *polarisation* dipole. The condition of static equilibrium gives the polarisability as:

$$\alpha = q_s^2/k \tag{2.238}$$

where q_s is the shell charge and k is the force constant of the harmonic spring.

In the adiabatic method, a fraction of the atomic mass is assigned to the shell to permit a dynamical description. The fraction of mass is chosen to ensure that the natural frequency of vibration ν of the harmonic spring (i.e.

$$\nu = \frac{1}{2\pi}\left[\frac{k}{x(1-x)m}\right]^{1/2}, \tag{2.239}$$

with m the atomic mass,) is well above the frequency of vibration of the whole atom in the bulk system. Dynamically the core-shell unit resembles a diatomic molecule with a harmonic bond, however the high vibrational frequency of the bond prevents effective exchange of kinetic energy between the core-shell unit and the remaining system. Therefore, from an initial condition in which the core-shell units have negligible internal vibrational energy, the units will remain close to this condition throughout the simulation. This is essential if the core shell unit is to maintain a net polarisation. (In practice there is a slow leakage of kinetic energy into the core-shell units, but this should should not amount to more than a few percent of the total kinetic energy.)

The calculation of the virial and stress tensor in this model is based on that for a diatomic molecule with charged atoms. The electrostatic and short ranged forces are calculated as described above. The forces of the harmonic springs are calculated as described for intramolecular harmonic bonds. The relationship between the kinetic energy and the temperature is different however, as the core-shell unit is permitted only three translational degrees of freedom, and the degrees of freedom corresponding to rotation and vibration of the unit are discounted (the kinetic energy of these is regarded as zero).

In DL_POLY Classic the shell forces are handled by the routine SHLFRC. The kinetic energy is calculated by CORSHL and the routine SHQNCH performs the temperature scaling. The dynamical shell model is used in conjunction with the methods for long ranged forces described above.

2.4.11 Relaxed Shell Model

The relaxed shell model is based on the same electrostatic principles as the dynamical shell model but in this case the shell is assigned a zero mass. This means the shell cannot be driven dynamically and instead the procedure is first to relax the shell to a condition of zero (or at least negligible) force at the start of the integration of the atomic motion and then integrate the motion of the finite mass core by conventional molecular dynamics. The relaxation of the shells in DL_POLY Classic is accomplished using conjugate gradients. Since each timestep of the algorithm entails a minimisation operation the cost per timestep for this algorithm is considerably more than the adiabatic shell model, however the integration timestep permitted is much larger (as much as a factor 10) so evolution through phase space is not necessarily very different in cost. A description of the method is presented in [50].

2.5 Integration algorithms

2.5.1 The Verlet Algorithms

DL_POLY integration algorithms are based on the Verlet scheme, which is both time reversible and simple [12]. It generates trajectories in the microcanonical (NVE) ensemble in which the total energy (kinetic plus potential energy) is conserved. If this property drifts or fluctuates excessively in the course of a simulation it indicates that the timestep is too large or the potential cutoffs too small (relative r.m.s. fluctuations in the total energy of 10^{-5} are typical with this algorithm).

DL_POLY Classic contains two versions of the Verlet algorithm. The first is the Verlet leapfrog (LF) algorithm and the second is the velocity Verlet (VV).

2.5.1.1 Verlet Leapfrog

The LF algorithm requires values of position ($\underline{r}$) and force ($\underline{f}$) at time t while the velocities ($\underline{v}$) are half a timestep behind. The first step is to advance the velocities to $t + (1/2)\Delta t$ by integration of the force:

$$\underline{v}(t + \frac{1}{2}\Delta t) \leftarrow \underline{v}(t - \frac{1}{2}\Delta t) + \Delta t \, \frac{\underline{f}(t)}{m} \tag{2.240}$$

where m is the mass of a site and Δt is the timestep.

The positions are then advanced using the new velocities:

$$\underline{r}(t+\Delta t) \leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t) \tag{2.241}$$

Molecular dynamics simulations normally require properties that depend on position and velocity *at the same time* (such as the sum of potential and kinetic energy). In the LF algorithm the velocity at time t is obtained from the average of the velocities half a timestep either side of time t:

$$\underline{v}(t) = \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t) + \underline{v}(t+\frac{1}{2}\Delta t)\right] \tag{2.242}$$

The full selection of LF integration algorithms within DL_POLY Classic is as follows:

NVE_1	Verlet leaprog with SHAKE
NVEQ_1	Rigid units with FIQA and SHAKE
NVEQ_2	Linked rigid units with QSHAKE
NVT_B1	Constant T (Berendsen [20]) with SHAKE
NVT_E1	Constant T (Evans [19]) with SHAKE
NVT_H1	Constant T (Hoover [21]) with SHAKE
NVTQ_B1	Constant T (Berendsen [20]) with FIQA and SHAKE
NVTQ_B2	Constant T (Berendsen [20]) with QSHAKE
NVTQ_H1	Constant T (Hoover [21]) with FIQA and SHAKE
NVTQ_H2	Constant T (Hoover [21]) with QSHAKE
NPT_B1	Constant T,P (Berendsen [20]) with FIQA and SHAKE
NPT_H1	Constant T,P+ (Hoover [21]) with SHAKE
NPTQ_B1	Constant T,P (Berendsen [20]) with FIQA and SHAKE
NPTQ_B2	Constant T,P (Berendsen [20]) with QSHAKE
NPTQ_H1	Constant T,P (Hoover [21]) with FIQA and SHAKE
NPTQ_H2	Constant T,P (Hoover [21]) with QSHAKE
NST_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with SHAKE
NST_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with SHAKE
NSTQ_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with FIQA and SHAKE
NSTQ_B2	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with QSHAKE
NSTQ_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with FIQA and SHAKE
NSTQ_H2	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with QSHAKE

In the above table the FIQA algorithm is Fincham's Implicit Quaternion Algorithm [15] and QSHAKE is the DL_POLY Classic algorithm combining rigid bonds and rigid bodies in the same molecule [17].

2.5.1.2 Velocity Verlet

The VV algorithm assumes that positions, velocities and forces are known at each full timestep. The algorithm proceeds in two stages as follows.

In the first stage a half step velocity is calculated:

$$\underline{v}(t+\frac{1}{2}\Delta t) \leftarrow \underline{v}(t) + \frac{1}{2}\Delta t\, \frac{\underline{f}(t)}{m} \tag{2.243}$$

and then the full timestep position is obtained:

$$\underline{r}(t+\Delta t) \leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t) \tag{2.244}$$

In the second stage, using the new positions, the next update of the forces $f(t+\Delta t)$ is obtained, from which the full step velocity is calculated using:

$$\underline{v}(t+\Delta t) \leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{1}{2}\Delta t\, \frac{\underline{f}(t+\Delta t)}{m} \tag{2.245}$$

Thus at the end of the two stages full synchronisation of the positions, forces and velocities is obtained.

The full selection of VV integration algorithms within DL_POLY Classic is as follows:

NVEVV_1	Velocity Verlet with RATTLE
NVEQVV_1	Rigid units with NOSQUISH and RATTLE
NVEQVV_2	Linked rigid units with QSHAKE
NVTVV_B1	Constant T (Berendsen [20]) with RATTLE
NVTVV_E1	Constant T (Evans [19]) with RATTLE
NVTVV_H1	Constant T (Hoover [21]) with RATTLE
NVTQVV_B1	Constant T (Berendsen [20]) with NOSQUISH and RATTLE
NVTQVV_B2	Constant T (Berendsen [20]) with QSHAKE
NVTQVV_H1	Constant T (Hoover [21]) with NOSQUISH and RATTLE
NVTQVV_H2	Constant T (Hoover [21]) with QSHAKE
NPTVV_B1	Constant T,P (Berendsen [20]) with NOSQUISH and RATTLE
NPTVV_H1	Constant T,P+ (Hoover [21]) with RATTLE
NPTQVV_B1	Constant T,P (Berendsen [20]) with NOSQUISH and RATTLE
NPTQVV_B2	Constant T,P (Berendsen [20]) with QSHAKE
NPTQVV_H1	Constant T,P (Hoover [21]) with NOSQUISH and RATTLE
NPTQVV_H2	Constant T,P (Hoover [21]) with QSHAKE
NSTVV_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with RATTLE
NSTVV_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with RATTLE
NSTQVV_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with NOSQUISH and RATTLE
NSTQVV_B2	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [20]) with QSHAKE
NSTQVV_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with NOSQUISH and RATTLE
NSTQVV_H2	Constant T,$\underline{\underline{\sigma}}$ (Hoover [21]) with QSHAKE

In the above table the NOSQUISH algorithm is the rotational algorithm of Miller *et al* [16] and QSHAKE is the DL_POLY Classic algorithm combining rigid bonds and rigid bodies in the same molecule [17].

2.5.1.3 Temperature and Energy Conservation

For both VV and LF the instantaneous temperature can be obtained from the atomic velocities assuming the system has no net momentum:

$$\mathcal{T} = \frac{\sum_{i=1}^{\mathcal{N}} m_i v_i^2(t)}{k_B f} \tag{2.246}$$

where i labels particles (which can be atoms or rigid molecules), $\mathcal{N}$ the number of particles in the system, k_B Boltzmanns constant and f the number of degrees of freedom in the system ($3\mathcal{N}-3$ if the system is periodic and without constraints).

The total energy of the system is a conserved quantity

$$\mathcal{H}_{\rm NVE} = U + KE \tag{2.247}$$

where U is the potential energy of the system and KE the kinetic energy at time t.

2.5.2 Bond Constraints

2.5.2.1 SHAKE

The SHAKE algorithm for bond constraints was devised by Ryckaert *et al.* [13] and is based on the Verlet leapfrog integration scheme [12]. It is a two stage scheme. In the first stage the leapfrog algorithm calculates the motion of the atoms in the system assuming a complete absence of the rigid bond forces. The positions of the atoms at the end of this stage do not conserve the distance constraint required by the rigid bond and a correction is necessary. In the second stage the deviation in the length of a given rigid bond is used retrospectively to compute the constraint force needed to conserve the bondlength. It is relatively simple to show that the constraint force has the form

$$\underline{G}_{ij} \approx \frac{\mu_{ij}(d_{ij}^2 - d_{ij}'^2)}{2\Delta t^2 \underline{d}_{ij}^o \cdot \underline{d}_{ij}'} \underline{d}_{ij}^o \tag{2.248}$$

where: μ_{ij} is the reduced mass of the two atoms connected by the bond; $\underline{d}_{ij}^o$ and $\underline{d}_{ij}'$ are the original and intermediate bond vectors; d_{ij} is the constrained bondlength; and Δt is the Verlet integration timestep. It should be noted that this formula is an approximation only.

For a system of simple diatomic molecules, computation of the constraint force will, in principle, allow the correct atomic positions to be calculated in one pass. However in the general polyatomic case this correction is merely an interim adjustment, not only because the above formula is approximate, but the successive correction of other bonds in a molecule has the effect of perturbing previously corrected bonds. The SHAKE algorithm is therfore iterative, with the correction cycle being repeated for all bonds until each has converged to the correct length, within a given tolerance. The tolerance may be of the order 10^{-4} Å to 10^{-8} Å depending on the precision desired.

The procedure may be summarised as follows:

1. All atoms in the system are moved using the Verlet algorithm, assuming an absence of rigid bonds (constraint forces). (This is stage 1 of the SHAKE algorithm.)

2. The deviation in each bondlength is used to calculate the corresponding constraint force (2.248) that (retrospectively) 'corrects' the bond length.

3. After the correction (2.248) has been applied to all bonds, every bondlength is checked. If the largest deviation found exceeds the desired tolerance, the correction calculation is repeated.

4. Steps 2 and 3 are repeated until all bondlengths satisfy the convergence criterion (This iteration constitutes stage 2 of the SHAKE algorithm).

DL_POLY Classic implements a parallel version of this algorithm [11] (see section 2.6.9). The subroutine NVE_1 implements the Verlet leapfrog algorithm with bond constraints for the NVE ensemble. The routine RDSHAKE_1 is called to apply the SHAKE corrections to position.

It should be noted that the fully converged constraint forces G_{ij} make a contribution to the system virial and the stress tensor.

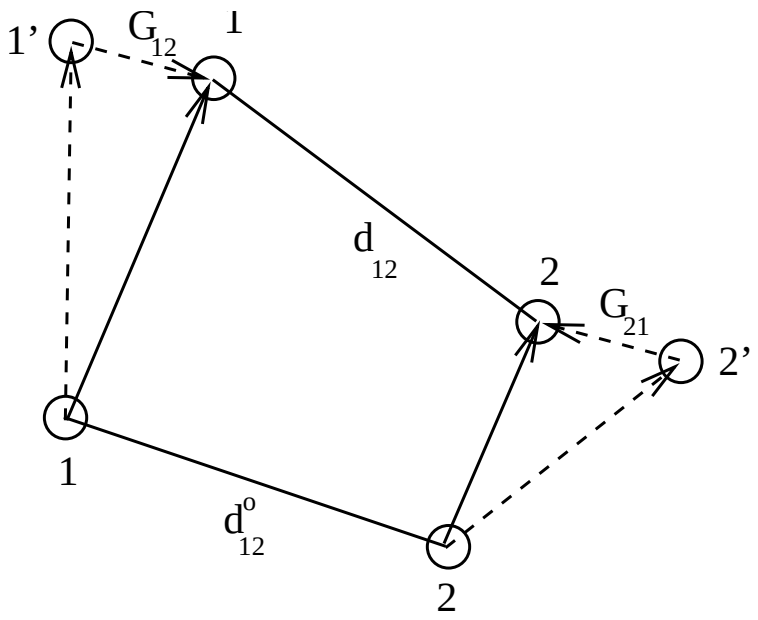

Figure 2.7: The SHAKE algorithm

The algorithm calculates the constraint force $\underline{G}_{12} = -\underline{G}_{21}$ that conserves the bondlength d_{12} between atoms 1 and 2, following the initial movement to positions $1'$ and $2'$ under the unconstrained forces $\underline{F}_1$ and $\underline{F}_2$.

The contribution to be added to the atomic virial (for each constrained bond) is

$$\mathcal{W} = -\underline{d}_{ij} \cdot \underline{G}_{ij}. \tag{2.249}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = d_{ij}^{\alpha} G_{ij}^{\beta}, \tag{2.250}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived from the pair forces is symmetric.

2.5.2.2 RATTLE

RATTLE [14] is the VV version of SHAKE. It has two parts: the first constrains the bondlength and the second adds an additional constaint to the velocities of the atoms in the constrained bond. The first of these constraints leads to an expression for the constriant force similar to that for SHAKE:

$$\underline{G}_{ij} \approx \frac{\mu_{ij}(d_{ij}^2 - d_{ij}'^2)}{\Delta t^2 \underline{d}_{ij}^o \cdot \underline{d}_{ij}'} \underline{d}_{ij}^o \tag{2.251}$$

Note that this formula differs from equation (2.248) by a factor of 2. This constraint force is applied during the first stage of the velocity Verlet algorithm.

The second constraint condition attempts to maintain the relative velocities of the atoms sharing a bond to a direction perpendicular to the bond vector. This provides another constraint force:

$$\underline{H}_{ij} \approx -\frac{2\mu_{ij}}{\Delta t}\frac{\underline{d}_{ij}\cdot(\underline{v}_j-\underline{v}_i)}{\underline{d}_{ij}^2}\underline{d}_{ij} \tag{2.252}$$

This constraint force is applied during the second stage of the velocity Verlet algorithm. Both constraint force calculations are iterative and are brought to convergence before proceeding to the next stage of the velocity Verlet scheme.

DL_POLY Classic implements a parallel version of RATTLE that is based on the same approach as SHAKE [11] (see section 2.6.9). The subroutine NVEVV_1 implements the velocity Verlet algorithm with bond constraints in the NVE ensemble. The subroutine RDRATTLE_R is called to apply the corrections to atom positions and the subroutine RDRATTLE_V is called to correct the atom velocities.

2.5.3 Potential of Mean Force (PMF) Constraints and the Evaluation of Free Energy

A generalization of bond constraints can be made to constrain a system to some point along a reaction coordinate. A simple example of such a reaction coordinate would be the distance between two ions in solution. If a number of simulations are conducted with the system constrained to different points along the reaction coordinate then the mean constraint force may be plotted as a function of reaction coordinate and the function integrated to obtain the free energy for the overall process [51]. The PMF constraint force, virial and contributions to the stress tensor are obtained in a manner analagous to that for a bond constraint (see previous section). The only difference is that the constraint is now applied between the centres of two groups which need not be atoms alone. DL_POLY Classic reports the PMF constraint virial, W, for each simulation. Users can convert this to the PMF constraint force from

$$G_{\rm PMF} = -\mathcal{W}_{\rm PMF}/d_{\rm PMF}$$

where $d_{\rm PMF}$ is the constraint distance between the two groups used to define the reaction coordinate.

DL_POLY Classic can calculate the PMF using either LF or VV algorithms. Subroutines PMFLF and PMF_SHAKE are used in the LF scheme and subroutines PMFVV, PMF_RATTLE_R and PMF_RATTLE_V are used in the VV scheme.

2.5.4 Thermostats

The system may be coupled to a heat bath to ensure that the average system temperature is maintained close to the requested temperature, $T_{\rm ext}$. When this is done the equations of motion are modified and the system no longer samples the microcanonical ensemble. Instead trajectories in the canonical (NVT) ensemble, or something close to it are generated. DL_POLY Classic comes with three different thermostats: Nosé -Hoover [21], Berendsen [20], and Gaussian constraints [19]. Of these only the Nosé-Hoover algorithm generates trajectories in the canonical (NVT) ensemble. The other methods will produce properties that typically differ from canonical averages by $\mathcal{O}(1/\mathcal{N})$ [12]

2.5.4.1 Nosé - Hoover Thermostat

In the Nosé-Hoover algorithm [21] Newton's equations of motion are modified to read:

$$\frac{d\underline{r}(t)}{dt} = \underline{v}(t)$$

$$\frac{d\underline{v}(t)}{dt} = \frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t) \tag{2.253}$$

$$\tag{2.254}$$

The friction coefficient, χ, is controlled by the first order differential equation

$$\frac{d\chi(t)}{dt} = \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\rm ext}) \tag{2.255}$$

where $Q = N_f k_B T_{\rm ext} \tau_T^2$ is the effective 'mass' of the thermoststat, τ_T is a specified time constant (normally in the range [0.5, 2] ps) and N_f is the number of degrees of freedom in the system. $\mathcal{T}(t)$ is the instantaneous temperature of the system at time t.

In the LF version of DL_POLY Classic χ is stored at half timesteps as it has dimensions of (1/time). The integration takes place as:

$$\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t-\frac{1}{2}\Delta t) + \Delta t \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\rm ext}) \\
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\frac{1}{2}\Delta t) + \chi(t+\frac{1}{2}\Delta t)\right] \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\frac{1}{2}\Delta t) + \Delta t\left[\frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t)\right] \\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t) + \underline{v}(t+\frac{1}{2}\Delta t)\right] \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t)
\end{aligned} \tag{2.256}$$

Since $\underline{v}(t)$ is required to calculate $\mathcal{T}(t)$ and itself, the algorithm requires several iterations to obtain self consistency. In DL_POLY Classic the number of iterations is set to 3 (4 if the system has bond constraints). The iteration procedure is started with the standard Verlet leapfrog prediction of $\underline{v}(t)$ and $\mathcal{T}(t)$. The conserved quantity is derived from the extended Hamiltonian for the system which, to within a constant, is the Helmholtz free energy:

$$\mathcal{H}_{\rm NVT} = U + KE + \frac{1}{2}Q\chi(t)^2 + \frac{Q}{\tau_T^2}\int_o^t \chi(s)ds \tag{2.257}$$

If bond constraints are present an extra iteration is required due to the call to the SHAKE routine. The algorithm is implemented in the DL_POLY routine NVT_H1, for systems with bond constraints.

In the VV version of DL_POLY Classic the Hoover algorithm is split into stages in accordance with the principles of Martyna *et al* [18] for designing reversible integrators. The scheme applied here is:

$$\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t) + \frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t) - T_{\rm ext}) \\
\underline{v}'(t) &\leftarrow \underline{v}(t) - \frac{\Delta t}{2}\chi(t+\frac{1}{2}\Delta t)\underline{v}(t) \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}'(t) + \frac{\Delta t}{2}\frac{\underline{f}(t)}{m} \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t) \\
& call \quad rattle(R)
\end{aligned}$$

$$\begin{aligned}
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m}\\
&\quad call \quad rattle(V)\\
\chi(t+\Delta t) &\leftarrow \chi(t+\frac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t)-T_{\rm ext})\\
\underline{v}(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t)-\frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}'(t+\Delta t)
\end{aligned} \tag{2.258}$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.251) and (2.252) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NVT}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V and NVTSCALE.

2.5.4.2 Berendsen Thermostat

In the Berendsen algorithm the instantaneous temperature is pushed towards the desired temperature by scaling the velocities at each step:

$$\chi(t) \leftarrow \left[1+\frac{\Delta t}{\tau_T}\left(\frac{T_{\rm ext}}{\mathcal{T}(t)}-1\right)\right]^{1/2} \tag{2.259}$$

The DL_POLY Classic LF routines implement this thermostat as follows.

$$\begin{aligned}
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \left[\underline{v}(t-\frac{1}{2}\Delta t)+\Delta t\frac{\underline{f}(t)}{m}\right]\chi(t)\\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t)+\underline{v}(t+\frac{1}{2}\Delta t)\right]\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\,\underline{v}(t+\frac{1}{2}\Delta t)
\end{aligned} \tag{2.260}$$

As with the Nosé-Hoover thermostat iteration is required to obtain self consistency of $\chi(t)$, $\underline{v}(t)$ and $\mathcal{T}(t)$, although it should be noted χ has different roles in the two thermostats. The Berendsen algorithm conserves total momentum but not energy. Here again the presence of constraint bonds requires an additional iteration with one application of SHAKE corrections. The algorithm is implemented in the DL_POLY routine NVT_B1, for systems including bond constraints.

The VV implementation of Berendsen's algorithm proceeds as folows:

$$\begin{aligned}
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}(t)+\frac{\Delta t}{2}\frac{\underline{f}(t)}{m}\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\,\underline{v}(t+\frac{1}{2}\Delta t)\\
&\quad call \quad rattle(R)\\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m}\\
&\quad call \quad rattle(V)\\
\chi &\leftarrow \left[1+\frac{\Delta t}{\tau_T}\left(\frac{\mathcal{T}}{T_{\rm ext}}-1\right)\right]^{1/2}\\
\underline{v}(t+\Delta t) &\leftarrow \chi\,\underline{v}'(t+\Delta t)
\end{aligned} \tag{2.261}$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.251) and (2.252) respectively. The integration is performed by the subroutine NVTVV_B1 which calls subroutines RATTLE_R and RATTLE_V.

2.5.5 Gaussian Constraints

Kinetic temperature can be made a constant of the equations of motion by imposing an additional constraint on the system. If one writes the equations of motions as :

$$\begin{aligned} \frac{d\underline{r}(t)}{dt} &= \underline{v}(t) \\ \frac{d\underline{v}(t)}{dt} &= \frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t) \end{aligned} \tag{2.262}$$

$$\tag{2.263}$$

with the temperature constraint

$$\frac{d\mathcal{T}}{dt} \propto \frac{d}{dt}\left(\sum_i (m_i v_i)^2\right) \propto \sum_i m_i^2 \underline{v}_i(t) \cdot \underline{f}_i(t) = 0 \tag{2.264}$$

then choosing

$$\chi = \frac{\sum_i m_i \underline{v}_i(t).\underline{f}_i(t)}{\sum_i m_i^2 v_i^2(t)} \tag{2.265}$$

minimises the "least squares" differences between the Newtonian and constrained trajectories.

Following Brown and Clarke [52] the algorithm is implemented in the LF scheme by calculating $\eta = 1/(1 + \chi\Delta t/2)$

$$\begin{aligned} \eta &\leftarrow \sqrt{\frac{T_{\rm ext}}{\mathcal{T}}} \\ \underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow (2\eta - 1)\underline{v}(t-\frac{1}{2}\Delta t) + \eta\Delta t\frac{\underline{f}(t)}{m} \\ \underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \end{aligned} \tag{2.266}$$

where $\mathcal{T}$ is obtained from standard Verlet leapfrog integration. Only one iteration is needed (two if the system has bond constraints) to constrain the instantaneous temperature to exactly $T_{\rm ext}$ however energy is not conserved by this algorithm. The algorithm is implemented in the DL_POLY routine NVT_E1 for systems with bond constraints.

The VV implementation of Evan's thermostat is as follows

$$\begin{aligned} \chi(t) &\leftarrow \sum_i m_i \underline{v}_i(t).\underline{f}_i(t) / \sum_i m_i^2 v_i^2(t) \\ \underline{v}'(t) &\leftarrow \underline{v}(t) - \frac{\Delta t}{2}\chi(t)\underline{v}(t) \\ \underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}'(t) + \frac{\Delta t}{2}\frac{\underline{f}(t)}{m} \\ \underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \\ & call\ \ rattle(R) \\ \underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m} \\ & call\ \ rattle(V) \\ \chi(t+\Delta t) &\leftarrow \sum_i m_i \underline{v}_i'(t+\Delta t).\underline{f}_i(t+\Delta t) / \sum_i m_i^2 v_i'^2(t+\Delta t) \\ \underline{v}(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t) - \frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}'(t+\Delta t) \end{aligned} \tag{2.267}$$

Routines *rattle*(R) and *rattle*(V) apply the bondlength and velocity constraint formulae (2.251) and (2.252) respectively. The integration is performed by the subroutine NVTVV_E1 which calls subroutines RATTLE_R and RATTLE_V.

2.5.6 Barostats

The size and shape of the simulation cell may be dynamically adjusted by coupling the system to a barostat in order to obtain a desired average pressure (P_{ext}) and/or isotropic stress tensor ($\underline{\underline{\sigma}}$). DL_POLY Classic has two such algorithms: a Hoover barostat and the Berendsen barostat. Only the former has a well defined conserved quantity.

2.5.6.1 The Hoover Barostat

DL_POLY Classic uses the Melchionna modification of the Hoover algorithm [53] in which the equations of motion couple a Nosé - Hoover thermostat and a barostat.

Cell size variation

For isotropic fluctuations the equations of motion are:

$$\begin{aligned}
\frac{d\underline{r}(t)}{dt} &= \underline{v}(t) + \eta(\underline{r}(t) - \underline{R}_0) \\
\frac{d\underline{v}(t)}{dt} &= \frac{\underline{\underline{f}}(t)}{m} - [\chi(t) + \eta(t)]\,\underline{v}(t) \\
\frac{d\chi(t)}{dt} &= \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\text{ext}}) + \frac{1}{Q}(W\eta(t)^2 - k_B T_{\text{ext}}) \\
\frac{d\eta(t)}{dt} &= \frac{3}{W}V(t)(\mathcal{P}(t) - P_{\text{ext}}) - \chi(t)\eta(t) \\
\frac{dV(t)}{dt} &= [3\eta(t)]V(t)
\end{aligned} \tag{2.268}$$

where $Q = N_f k_B T_{\text{ext}} \tau_T^2$ is the effective 'mass' of the thermostat and $W = N_f k_B T_{\text{ext}} \tau_P^2$ is the effective 'mass' of the barostat. N_f is the number of degrees of freedom, η is the barostat friction coefficient, R_0 the system centre of mass, τ_T and τ_P are specified time constants for temperature and pressure fluctuations respectively, $\mathcal{P}(t)$ is the instantaneous pressure and V the system volume.

The conserved quantity is, to within a constant, the Gibbs free energy of the system:

$$\mathcal{H}_{NPT} = U + KE + \mathcal{P}_{\text{ext}}V(t) + \frac{1}{2}Q\chi(t)^2 + \frac{1}{2}W\eta(t)^2 + \int_o^t (\frac{Q}{\tau_T^2}\chi(s) + k_B\mathcal{T}_{\text{ext}})ds \tag{2.269}$$

The algorithm is readily implemented in the LF scheme as:

$$\begin{aligned}
\chi(t+\tfrac{1}{2}\Delta t) &\leftarrow \chi(t-\tfrac{1}{2}\Delta t) + \frac{\Delta t N_f k_B}{Q}(\mathcal{T}(t) - T_{\text{ext}}) + \frac{\Delta t}{Q}(W\eta(t)^2 - k_B T_{\text{ext}}) \\
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\tfrac{1}{2}\Delta t) + \chi(t+\tfrac{1}{2}\Delta t)\right] \\
\eta(t+\tfrac{1}{2}\Delta t) &\leftarrow \eta(t-\tfrac{1}{2}\Delta t) + \Delta t\left\{\frac{3V(t)}{W}(\mathcal{P}(t) - P_{\text{ext}}) - \chi(t)\eta(t)\right\} \\
\eta(t) &\leftarrow \frac{1}{2}\left[\eta(t-\tfrac{1}{2}\Delta t) + \eta(t+\tfrac{1}{2}\Delta t)\right] \\
\underline{v}(t+\tfrac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\tfrac{1}{2}\Delta t) + \Delta t\left[\frac{\underline{\underline{f}}(t)}{m} - [\chi(t) + \eta(t)]\,\underline{v}(t)\right]
\end{aligned}$$

$$\begin{aligned}
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t)+\underline{v}(t+\frac{1}{2}\Delta t)\right]\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\left(\underline{v}(t+\frac{1}{2}\Delta t)+\eta(t+\frac{1}{2}\Delta t)\left[\underline{r}(t+\frac{1}{2}\Delta t)-\underline{R}_0\right]\right)\\
\underline{r}(t+\frac{1}{2}\Delta t) &\leftarrow \frac{1}{2}\left[\underline{r}(t)+\underline{r}(t+\Delta t)\right]
\end{aligned} \tag{2.270}$$

Like the LF Nosé-Hoover thermostat, several iterations are required to obtain self consistency. DL_POLY Classic uses 4 iterations (5 if bond constraints are present) with the standard Verlet leapfrog predictions for the initial estimates of $\mathcal{T}(t)$, $\mathcal{P}(t)$, $\underline{v}(t)$ and $\underline{r}(t+\frac{1}{2}\Delta t)$. Note also that the change in box size requires the SHAKE algorithm to be called each iteration with the new cell vectors and volume obtained from:

$$\begin{aligned}
V(t+\Delta t) &\leftarrow V(t)\exp\left[3\Delta t\;\eta(t+\frac{1}{2}\Delta t)\right]\\
\underline{\underline{\mathbf{H}}}(t+\Delta t) &\leftarrow \exp\left[\Delta t\;\eta(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t)
\end{aligned} \tag{2.271}$$

where $\underline{\underline{\mathbf{H}}}$ is the cell matrix whose columns are the three cell vectors $\underline{a},\underline{b},\underline{c}$.

The isotropic changes to cell volume are implemented in the DL_POLY LF routine NPT_H1 which allows for systems containing bond constraints.

The implementation in the VV algorithm follows the scheme:

$$\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{2Q}(W\eta(t)^2-k_BT_{\rm ext})\\
\underline{v}'(t) &\leftarrow \underline{v}(t)-\frac{\Delta t}{2}\chi(t+\frac{1}{2}\Delta t)\underline{v}(t)\\
\eta(t+\frac{1}{2}\Delta t) &\leftarrow \eta(t)+\frac{\Delta t}{2}\left\{\frac{3V(t)}{W}(\mathcal{P}(t)-P_{\rm ext})-\chi(t)\eta(t)\right\}\\
\underline{v}''(t) &\leftarrow \underline{v}'(t)-\frac{\Delta t}{2}\eta(t+\frac{1}{2}\Delta t)\underline{v}'(t)\\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}''(t)+\frac{\Delta t}{2}\frac{\underline{f}(t)}{m}\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\;\underline{v}(t+\frac{1}{2}\Delta t)\\
&\quad call\quad rattle(R)\\
V(t+\Delta t) &\leftarrow V(t)\exp\left[3\Delta t\;\eta(t+\frac{1}{2}\Delta t)\right]\\
\underline{\underline{\mathbf{H}}}(t+\Delta t) &\leftarrow \exp\left[\Delta t\;\eta(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t)\\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m}\\
&\quad call\quad rattle(V)\\
\eta(t+\Delta t) &\leftarrow \eta(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\left\{\frac{V(t+\Delta t)}{W}(\mathcal{P}(t+\Delta t)-P_{\rm ext})-\chi(t+\Delta t)\eta(t+\Delta t)\right\}\\
\underline{v}''(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t)-\frac{\Delta t}{2}\eta(t+\Delta t)\underline{v}'(t+\Delta t)\\
\chi(t+\Delta t) &\leftarrow \chi(t+\frac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t)-T_{\rm ext})+\frac{\Delta t}{2Q}(W\eta(t+\Delta t)^2-k_BT_{\rm ext})\\
\underline{v}(t+\Delta t) &\leftarrow \underline{v}''(t+\Delta t)+\frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}''(t+\Delta t)
\end{aligned} \tag{2.272}$$

Routines *rattle*(R) and *rattle*(V) apply the bondlength and velocity constraint formulae (2.251) and (2.252) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NPT}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V, NPTSCALE_T and NPTSCALE_P.

Cell size and shape variation

The isotropic algorithms may be extended to allowing the cell shape to vary by defining η as a tensor, $\underline{\underline{\eta}}$.

The LF equations of motion are implemented as:

$$\begin{aligned}
\chi(t+\tfrac{1}{2}\Delta t) &\leftarrow \chi(t-\tfrac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{Q}(WTr(\underline{\underline{\eta}}(t))^2-9k_BT_{\rm ext})\\
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\tfrac{1}{2}\Delta t)+\chi(t+\tfrac{1}{2}\Delta t)\right]\\
\underline{\underline{\eta}}(t+\tfrac{1}{2}\Delta t) &\leftarrow \underline{\underline{\eta}}(t-\tfrac{1}{2}\Delta t)+\frac{\Delta t V(t)}{W}\left(\underline{\underline{\sigma}}(t)-P_{\rm ext}\underline{\underline{\mathbf{1}}}\right)-\Delta t\chi(t)\underline{\underline{\eta}}(t)\\
\underline{\underline{\eta}}(t) &\leftarrow \frac{1}{2}\left[\underline{\underline{\eta}}(t-\tfrac{1}{2}\Delta t)+\underline{\underline{\eta}}(t+\tfrac{1}{2}\Delta t)\right]\\
\underline{v}(t+\tfrac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\tfrac{1}{2}\Delta t)+\Delta t\left[\frac{\underline{\underline{f}}(t)}{m}-\left[\chi(t)\underline{\underline{\mathbf{1}}}+\underline{\underline{\eta}}(t)\right]\underline{v}(t)\right]\\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\tfrac{1}{2}\Delta t)+\underline{v}(t+\tfrac{1}{2}\Delta t)\right]\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\left(\underline{v}(t+\tfrac{1}{2}\Delta t)+\underline{\underline{\eta}}(t+\tfrac{1}{2}\Delta t)\left[\underline{r}(t+\tfrac{1}{2}\Delta t)-\underline{R}_0\right]\right)\\
\underline{r}(t+\tfrac{1}{2}\Delta t) &\leftarrow \frac{1}{2}\left[\underline{r}(t)+\underline{r}(t+\Delta t)\right]
\end{aligned}\tag{2.273}$$

where $\underline{\underline{\mathbf{1}}}$ is the identity matrix and $\underline{\underline{\sigma}}$ the pressure tensor. The new cell vectors are calculated from

$$\underline{\underline{\mathbf{H}}}(t+\Delta t) \leftarrow \exp\left[\Delta t\,\underline{\underline{\eta}}(t+\tfrac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t)\tag{2.274}$$

DL_POLY Classic uses a power series expansion truncated at the quadratic term to approximate the exponential of the tensorial term. The new volume is found from

$$V(t+\Delta t)\leftarrow V(t)\exp\left[\Delta t\,Tr(\underline{\underline{\eta}})\right]\tag{2.275}$$

The conserved quantity is

$$\mathcal{H}_{NST}=U+KE+\mathcal{P}_{\rm ext}V(t)+\frac{1}{2}Q\chi(t)^2+\frac{1}{2}WTr(\underline{\underline{\eta(\mathbf{t})}})^2+\int_o^t\chi(s)(\frac{Q}{\tau_T^2}+9k_B\mathcal{T}_{\rm ext})ds\tag{2.276}$$

This algorithm is implemented in the routine NST_H1, with bond constraints.

The VV version of this algorithm is implemented as:

$$\begin{aligned}
\chi(t+\tfrac{1}{2}\Delta t) &\leftarrow \chi(t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{2Q}(WTr(\underline{\underline{\eta}}(t))^2-9k_BT_{\rm ext})\\
\underline{v}'(t) &\leftarrow \underline{v}(t)-\frac{\Delta t}{2}\chi(t+\tfrac{1}{2}\Delta t)\underline{v}(t)\\
\underline{\underline{\eta}}(t+\tfrac{1}{2}\Delta t) &\leftarrow \underline{\underline{\eta}}(t)+\frac{\Delta t}{2}\left\{\frac{V(t)}{W}(\underline{\underline{\sigma}}(t)-P_{\rm ext}\underline{\underline{\mathbf{1}}})-\chi(t)Tr(\underline{\underline{\eta}}(t))\right\}
\end{aligned}$$

$$
\begin{aligned}
\underline{v}''(t) &\leftarrow \underline{v}'(t) - \frac{\Delta t}{2}\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)\underline{v}'(t) \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}''(t) + \frac{\Delta t}{2}\frac{\underline{f}(t)}{m} \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \\
&\ call\ \ rattle(R) \\
V(t+\Delta t) &\leftarrow V(t)\exp\left[3\Delta t\ \eta(t+\frac{1}{2}\Delta t)\right] \\
\underline{\underline{\mathbf{H}}}(t+\Delta t) &\leftarrow \exp\left[\Delta t\ \eta(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t) \\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m} \\
&\ call\ \ rattle(V) \\
\underline{\underline{\eta}}(t+\Delta t) &\leftarrow \underline{\underline{\eta}}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\left\{\frac{V(t+\Delta t)}{W}(\underline{\underline{\eta}}(t+\Delta t) - P_{\rm ext}\underline{\underline{\mathbf{1}}}) - \chi(t+\Delta t)Tr(\underline{\underline{\eta}}(t+\Delta t))\right\} \\
\underline{v}''(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t) - \frac{\Delta t}{2}\underline{\underline{\eta}}(t+\Delta t)\underline{v}'(t+\Delta t) \\
\chi(t+\Delta t) &\leftarrow \chi(t+\frac{1}{2}\Delta t) + \frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t) - T_{\rm ext}) + \frac{\Delta t}{2Q}(WTr(\underline{\underline{\eta}}(t+\Delta t))^2 - 9k_B T_{\rm ext}) \\
\underline{v}(t+\Delta t) &\leftarrow \underline{v}''(t+\Delta t) + \frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}''(t+\Delta t)
\end{aligned}
\tag{2.277}
$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.251) and (2.252) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NST}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V, NSTSCALE_T and NSTSCALE_P.

2.5.6.2 Berendsen Barostat

With the Berendsen barostat the system is made to obey the equation of motion

$$\frac{d\mathcal{P}}{dt} = (P_{\rm ext} - \mathcal{P})/\tau_P \tag{2.278}$$

Cell size variations

In the isotropic implementation, at each step the MD cell volume is scaled by by a factor η and the coordinates, and cell vectors, by $\eta^{1/3}$ where

$$\eta = 1 - \frac{\beta\Delta t}{\tau_P}(P_{\rm ext} - \mathcal{P}) \tag{2.279}$$

and β is the isothermal compressibility of the system. The Berendesen thermostat is applied at the same time. In practice β is a specified constant which DL_POLY Classic takes to be the isothermal compressibility of liquid water. The exact value is not critical to the algorithm as it relies on the ratio τ_P/β. τ_P is specified by the user.

The LF version of this algorithm is implemented in NPT_B1 with 4 or 5 iterations used to obtain self consistency in the $\underline{v}(t)$. It calls RDSHAKE_1 to handle constraints. The VV version is implemented in subroutine NVTVV_B1, which calls constraint subroutines RATTLE_R and RATTLE_V.

Cell size and shape variations

The extension of the isotropic algorithm to anisotropic cell variations is straightforward. The tensor $\underline{\underline{\eta}}$ is defined by

$$\underline{\underline{\eta}} = \underline{\underline{\mathbf{1}}} - \frac{\beta \Delta t}{\tau_P}(P_{\text{ext}}\underline{\underline{\mathbf{1}}} - \underline{\underline{\sigma}}) \tag{2.280}$$

and the new cell vectors given by

$$\underline{\underline{\mathbf{H}}}(t + \Delta t) \leftarrow \underline{\underline{\eta}}\,\underline{\underline{\mathbf{H}}}(t) \tag{2.281}$$

As in the isotropic case the Berendsen thermostat is applied simultaneously and 4 or 5 iterations are used to obtain convergence. The LF version of the algorithm is implemented in subroutine NST_B1 and the VV version in NSTVV_B1. The former calls RDSHAKE_1 to handle constraints and the latter calls subroutines RATTLE_R and RATTLE_V.

2.5.7 Rigid Bodies and Rotational Integration Algorithms

2.5.7.1 Description of Rigid Body Units

A rigid body unit is a collection of point atoms whose local geometry is time invariant. One way to enforce this in a simulation is to impose a sufficient number of bond constraints between the atoms in the unit. However, in many cases this is may be either problematic or impossible. Examples in which it is impossible to specify sufficient bond constraints are

1. linear molecules with more than 2 atoms (e.g. CO_2)
2. planar molecules with more than three atoms (e.g. benzene).

Even when the structure *can* be defined by bond constraints the network of bonds produced may be problematic. Normally, they make the iterative SHAKE procedure slow, particularly if a ring of constraints is involved (as occurs when one defines water as a constrained triangle). It is also possible, inadvertently, to over constrain a molecule (e.g. by defining a methane tetrahedron to have 10 rather than 9 bond constraints) in which case the SHAKE procedure will become unstable. In addition, massless sites (e.g. charge sites) cannot be included in a simple constraint approach making modelling with potentials such as TIP4P water impossible.

All these problems may be circumvented by defining rigid body units, the dynamics of which may be described in terms of the translational motion of the center of mass (COM) and rotation about the COM. To do this we need to define the appropriate variables describing the position, orientation and inertia of a rigid body, and the rigid body equations of motion. [5]

The mass of a rigid unit M is the sum of the atomic masses in that unit:

$$M = \sum_{j=1}^{N_{sites}} m_j. \tag{2.282}$$

where m_j is the mass of an atom and the sum includes all sites (N_{sites}) in the body. The position of the rigid unit is defined as the location of its centre of mass $\underline{R}$:

$$\underline{R} = \frac{1}{M} \sum_{j=1}^{N_{sites}} m_j \underline{r}_j, \tag{2.283}$$

[5] An alternative approach is to define "basic" and "secondary" particles. The basic particles are the minimun number needed to define a local body axis system. The remaining particle positions are expressed in terms of the COM and the basic particles. Ordinary bond constraints can then be applied to the basic particles provided the forces and torques arising from the secondary particles are transferred to the basic particles in a physically meaningful way.

where $\underline{r}_j$ is the position vector of atom j. The rigid body translational velocity $\underline{V}$ is defined by:

$$\underline{V} = \frac{1}{M} \sum_{j=1}^{N_{sites}} m_j \underline{v}_j, \tag{2.284}$$

where $\underline{v}_j$ is the velocity of atom j. The net translational force acting on the rigid body unit is the vector sum of the forces acting on the atoms of the body:

$$\underline{F} = \sum_{j=1}^{N_{sites}} \underline{f}_j \tag{2.285}$$

where $\underline{f}_j$ is the force on a rigid unit site

A rigid body also has associated with it a rotational inertia matrix $\underline{\underline{\mathbf{I}}}$, whose components are given by

$$I_{\alpha\beta} = \sum_{j}^{N_{sites}} m_j (d_j^2 \delta_{\alpha\beta} - d_j^\alpha r_j^\beta) \tag{2.286}$$

where $\underline{d}_j$ is the displacement vector of the atom j from the COM, and is given by:

$$\underline{d}_j = \underline{r}_j - \underline{R}. \tag{2.287}$$

It is common practice in the treatment of rigid body motion to define the position $\underline{R}$ of the body in a universal frame of reference (the so called laboratory or inertial frame), but to describe the moment of inertia tensor in a frame of reference that is localised in the rigid body and changes as the rigid body rotates. Thus the local body frame is taken to be that in which the rotational inertia tensor $\underline{\underline{\hat{\mathbf{I}}}}$ is diagonal and the components satisfy $I_{xx} \geq I_{yy} \geq I_{zz}$. In this local frame (the so called *Principal Frame*) the inertia tensor is therefore constant.

The orientation of the local body frame with respect to the space fixed frame is described via a four dimensional unit vector, the quaternion

$$\underline{q} = [q_0, q_1, q_2, q_3]^T, \tag{2.288}$$

and the rotational matrix $\underline{\underline{\mathbf{R}}}$ to transform from the local body frame to the space fixed frame is the unitary matrix

$$\underline{\underline{\mathbf{R}}} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{2.289}$$

so that if $\hat{\underline{d}}_j$ is the position of an atom in the local body frame (with respect to its COM), its position in the universal frame (w.r.t. its COM) is given by

$$\underline{d}_j = \underline{\underline{\mathbf{R}}}\, \hat{\underline{d}}_j \tag{2.290}$$

With these variables defined we can now consider the equations of motion for the rigid body unit.

2.5.7.2 Integration of the Rigid Body Equations of Motion

The equations of translational motion of a rigid body are the same as those describing the motion of a single atom, except that the force is the total force acting on the rigid body i.e. $\underline{F}$ in equation (2.285) and the mass is the total mass of the rigid body unit i.e. M in equation (2.282). These

equations can be integrated by the standard Verlet LF or VV algorithms described in the previous sections. Thus we need only consider the rotational motion here.

The rotational equation of motion for a rigid body is

$$\underline{\tau} = \frac{d}{dt}\underline{J} = \frac{d}{dt}\left(\underline{\underline{\mathbf{I}}}\underline{\omega}\right), \tag{2.291}$$

in which $\underline{J}$ is the angular momentum of the rigid body defined by the expression

$$\underline{J} = \sum_{j=1}^{N_{sites}} m_j \underline{d}_j \times \underline{v}_j, \tag{2.292}$$

and $\underline{\omega}$ is the angular velocity.

The vector $\underline{\tau}$ is the torque acting on the body in the universal frame and is given by

$$\underline{\tau} = \sum_{j=1}^{N_{sites}} \underline{d}_j \times \underline{f}_j. \tag{2.293}$$

The rotational equations of motion, written in the local frame of the rigid body, are given by Euler's equations

$$\begin{aligned} \dot{\hat{\omega}}_x &= \frac{\hat{\tau}_x}{\hat{I}_{xx}} + (\hat{I}_{yy} - \hat{I}_{zz})\hat{\omega}_y\hat{\omega}_z \\ \dot{\hat{\omega}}_y &= \frac{\hat{\tau}_y}{\hat{I}_{yy}} + (\hat{I}_{zz} - \hat{I}_{xx})\hat{\omega}_z\hat{\omega}_z \\ \dot{\hat{\omega}}_y &= \frac{\hat{\tau}_z}{\hat{I}_{zz}} + (\hat{I}_{xx} - \hat{I}_{yy})\hat{\omega}_x\hat{\omega}_y \end{aligned} \tag{2.294}$$

The vector $\underline{\hat{\omega}}$ is the angular velocity transformed to the local body frame. Integration of $\hat{\omega}$ is complicated by the fact that as the rigid body rotates , so does the local reference frame. So it is necessary to integrate equations (2.294) simultaneously with an integration of the quaternions describing the orientiation of the rigid body. The equation describing this is:

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}\begin{pmatrix} 0 \\ \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \end{pmatrix} \tag{2.295}$$

Rotational motion in DL_POLY Classic is handled by two different methods. For LF implementation, the Fincham Implicit Quaternion Algorithm (FIQA) is used [15]. The VV implementation uses the NOSQUISH algorithm of Miller *et al.* [16].

The LF implementation begins by integrating the angular velocity equation in the local frame.

$$\hat{\omega}(t + \frac{\Delta t}{2}) = \hat{\omega}(t - \frac{\Delta t}{2}) + \Delta t\, \dot{\underline{\hat{\omega}}}(t) \tag{2.296}$$

The new quaternions are found using the FIQA algorithm. In this algorithm the new quaternions are found by solving the implicit equation

$$\underline{q}(t + \Delta t) = \underline{q}(t) + \frac{\Delta t}{2}\left(\underline{\underline{\mathbf{Q}}}[\underline{q}(t)]\underline{\hat{w}}(t) + \underline{\underline{\mathbf{Q}}}[\underline{q}(t + \Delta t)]\underline{\hat{w}}(t + \Delta t)\right) \tag{2.297}$$

where $\underline{\hat{w}} = [0, \hat{\omega}]^T$ and $\underline{\underline{\mathbf{Q}}}[\underline{q}]$ is

$$\underline{\underline{\mathbf{Q}}} = \frac{1}{2}\begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & -q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \tag{2.298}$$

The above equation is solved iteratively with

$$\underline{q}(t + \Delta t) = \underline{q}(t) + \Delta t \; \underline{\underline{\mathbf{Q}}}[\underline{q}(t)]\underline{\hat{w}}(t) \tag{2.299}$$

as the first guess. Typically no more than 3 or 4 iterations are needed for convergence. At each step the constraint

$$\|\underline{q}(t + \Delta t)\| = 1 \tag{2.300}$$

is imposed.

The NVE LF algorithm is implemented in NVEQ_1 which allows for a system containing a mixture of rigid bodies and atomistic species, provided the rigid bodies are not linked to other species by constraint bonds.

The VV implementation is based on the NOSQUISH algorithm of Miller *et al.* [16]. In addition to the quaternions it requires quaternion momenta defined by

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = 2\begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}\begin{pmatrix} 0 \\ \hat{I}_{xx}\hat{\omega}_x \\ \hat{I}_{yy}\hat{\omega}_y \\ \hat{I}_{zz}\hat{\omega}_z \end{pmatrix} \tag{2.301}$$

and quaternion torques defined by

$$\begin{pmatrix} \Upsilon_0 \\ \Upsilon_1 \\ \Upsilon_2 \\ \Upsilon_3 \end{pmatrix} = 2\begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}\begin{pmatrix} 0 \\ \hat{\tau}_x \\ \hat{\tau}_y \\ \hat{\tau}_z \end{pmatrix} \tag{2.302}$$

(It should be noted that vectors $\underline{p}$ and $\underline{\Upsilon}$ are 4-component vectors.) The quaternion momenta are first updated a half-step using the formula

$$\underline{p}(t + \frac{\Delta t}{2}) \leftarrow \underline{p}(t) + \frac{\Delta t}{2}\underline{\Upsilon}(t) \tag{2.303}$$

Next a sequence of operations is applied to the quaternions and the quaternion momenta in the order

$$e^{i\mathcal{L}_3(\delta t/2)}e^{i\mathcal{L}_2(\delta t/2)}e^{i\mathcal{L}_1(\delta t)}e^{i\mathcal{L}_2(\delta t/2)}e^{i\mathcal{L}_3(\delta t/2)} \tag{2.304}$$

which preserves the symplecticness of the operations (see reference [18]). Note that δt is some submultiple of Δt. (In DL_POLY Classic the default is $\Delta t = 10\delta t$.) The operators themselves are of the following kind:

$$\begin{aligned} e^{i\mathcal{L}(\delta t)}\underline{q} &= cos(\zeta_k \delta t)\underline{q} + sin(\zeta_k \delta t)P_k\underline{q} \\ e^{i\mathcal{L}(\delta t)}\underline{p} &= cos(\zeta_k \delta t)\underline{p} + sin(\zeta_k \delta t)P_k\underline{p} \end{aligned} \tag{2.305}$$

where P_k is a permutation operator with $k = 0, \ldots, 3$ with the following properties

$$\begin{aligned} P_0\underline{q} &= \{q_0, q_1, q_2, q_3\} \\ P_1\underline{q} &= \{-q_1, q_0, q_3, -q_2\} \\ P_2\underline{q} &= \{-q_2, -q_3, q_0, q_1\} \\ P_3\underline{q} &= \{-q_3, q_2, -q_1, q_0\} \end{aligned} \tag{2.306}$$

and the angular velocity ζ_k is defined as

$$\zeta_k = \frac{1}{4I_k}\underline{p}^T P_k \underline{q}. \tag{2.307}$$

Equations (2.304) to (2.306) represent the heart of the NOSQUISH algorithm and are repeatedly applied (10 times in DL_POLY Classic). The final result is the quaternion updated to the full timestep value i.e. $\underline{q}(t+\Delta t)$. These equations form part of the first stage of the VV algorithm.

In the second stage of the VV algorithm, new torques are used to update the quaternion momenta to a full timestep.

$$\underline{p}(t+\Delta t) \leftarrow \underline{p}(t+\frac{\Delta t}{2}) + \frac{\Delta t}{2}\underline{\Upsilon}(t+\Delta t) \tag{2.308}$$

The NVE implementation of this algorithm is in the subroutine NVEQVV_1 which calls the NOSQUISH subroutine to perform the rotation operation. The subroutine also calls RATTLE_R and RATTLE_V to handle any rigid bonds which may be present.

Thermostats and Barostats

It is straightforward to couple the rigid body equations of motion to a thermostat and/or barostat. The thermostat is coupled to both the translational and rotational degrees of freedom and so both the translational and rotational velocities are propagated in an analogous manner to the thermostated atomic velocities. The barostat, however, is coupled only to the translational degrees of freedom and not to the rotation.DL_POLY Classic supports both Hoover and Berendsen thermostats and barostats for systems containing rigid bodies.

For LF integration the Hoover thermostat is implemented in NVTQ_H1, the Hoover isotropic barostat (plus thermostat) in NPTQ_H1 and the anisotropic barostat in NSTQ_H1. The analogous routines for the Berendsen algorithms are NVTQ_B1, NPTQ_B1 and NSTQ_B1. These subroutines also call RDSHAKE_1 to handle any rigid bonds which may be present.

For VV integration the Hoover thermostat is implemented in NVTQVV_H1 (NVTQSCL) , the Hoover isotropic barostat (plus thermostat) in NPTQVV_H1 (NPTQSCL_T, NPTQSCL_P) and the anisotropic barostat in NSTQVV_H1 (NSTQSCL_T, NSTQSCL_P). The analogous routines for the Berendsen algorithms are NVTQVV_B1, NPTQVV_B1 and NSTQVV_B1. (The subroutines in brackets represent supporting subroutines.) These subroutines also call RATTLE_R and RATTLE_V to handle any rigid bonds which may be present.

2.5.7.3 Linked Rigid Bodies

The above integration algorithms can be used for rigid bodies in systems containing "atomic" species (whose equations of motion are integrated with the standard leapfrog algorithm). These rigid bodies may even be linked to other species (including other rigid bodies) by extensible bonds. However if a rigid body is linked to an atom or another rigid body by a bond *constraint* the above algorithms are not adequate. The reason is that the constraint will introduce an additional force and torque on the body that can only be found *after* the integration of the unconstrained unit. DL_POLY Classic has a suite of integration algorithms to cope with this situation in which both the constraint conditions and the quaternion equations are solved similtaneously using an extension of the SHAKE algorithm called "QSHAKE" [17]. It has been cast in both LF and VV forms. We will describe here how it works for VV, the LF version is decribed in [17].

Firstly we assume a rigid body (A) is connected to another (B) at timestep $t_n = n\Delta t$ via bonds between atoms at positions $\underline{r}^n_{Ap}$ and $\underline{r}^n_{Bp}$ given by

$$\begin{aligned} \underline{r}^n_{Ap} &= \underline{R}^n_A + \underline{d}^n_{Ap} \\ \underline{r}^n_{Bp} &= \underline{R}^n_B + \underline{d}^n_{Bp} \end{aligned} \tag{2.309}$$

where $\underline{R}$ represents the rigid body COM and $\underline{d}$ the displacement of the atom from the relevant COM. The subscript p indicates that these are the atoms providing the links.

In the first stage of the VV QSHAKE algorithm, the rigid bodies are allowed to move unrestricted. Our task is then to find the the constraint force $\underline{G}^n_{AB}$ which would preserve the constraint bondlength i.e. $d^n_{ABp} = d^{n+1}_{ABp}$. Assuming we know this force we can write:

$$\underline{R}^{n+1}_A = \underline{\tilde{R}}^{n+1}_A + \frac{\Delta t^2}{2M_A}\underline{G}^n_{AB} \tag{2.310}$$

in which the *tilde* $\tilde{x}$ indicates the corresponding variable computed in the absence of the constraint force. (For brevity, in this and subsequent equations we leave out corresponding equations for body B.)

We can also write the true torque at timestep t_n (i.e. $\underline{\tau}^n$) as

$$\underline{\tau}^n = \underline{\tilde{\tau}}^n + d^n_{Ap} \times \underline{G}^n_{AB}. \tag{2.311}$$

It may be easily shown from this and equation (2.291) that

$$\underline{\dot{\omega}}^n_A = \underline{\dot{\tilde{\omega}}}^n_A + \left(\underline{\underline{\mathbf{I}}}^n\right)^{-1}\left(d^n_{Ap} \times \underline{G}^n_{AB}\right) \tag{2.312}$$

from which it follows that

$$\underline{d}^{n+1}_{Ap} = \underline{\tilde{d}}^{n+1}_{Ap} + \frac{\Delta t^2}{2} g^n_{AB}\underline{U}^n_A \times \underline{d}^n_{Ap} \tag{2.313}$$

where we have defined

$$\underline{U}^n_A = \left(\underline{\underline{\mathbf{I}}}^n_A\right)^{-1}\left(\underline{d}^n_{Ap} \times \underline{d}^n_{ABp}\right) \tag{2.314}$$

and we have used the identity

$$\underline{G}^n_{AB} = g^n_{AB}\underline{d}^n_{ABp}$$

where g^n_{AB} is a scalar quantity. Now, the true position (at timestep t_{n+1}) of the link atom on rigid body A is

$$\underline{r}^{n+1}_{Ap} = \underline{R}^{n+1}_A + \underline{d}^{n+1}_{Ap} \tag{2.315}$$

and inserting (2.310) and (2.313) leads to

$$\underline{r}^{n+1}_{Ap} = \underline{\tilde{R}}^{n+1}_A + \underline{\tilde{d}}^{n+1}_{Ap} + g^n_{AB}\frac{\Delta t^2}{2}\underline{\Theta}_A \tag{2.316}$$

where

$$\underline{\Theta}_A = \left(\frac{\underline{d}^n_{ABp}}{M_A} + \underline{U}^n_A \times \underline{d}^n_{Ap}\right). \tag{2.317}$$

Since $\underline{d}^{n+1}_{ABp} = \underline{r}^{n+1}_{Ap} - \underline{r}^{n+1}_{Bp}$ we can easily obtain

$$\underline{d}^{n+1}_{ABp} = \underline{\tilde{d}}^{n+1}_{ABp} + g^n_{AB}\frac{\Delta t^2}{2}(\underline{\Theta}_A - \underline{\Theta}_B) \tag{2.318}$$

Squaring both sides and neglecting terms of order higher than $O(\Delta t^2)$ gives after rearrangement:

$$g^n_{AB} \approx \frac{(d^{n+1}_{ABp})^2 - (\tilde{d}^{n+1}_{ABp})^2}{\Delta t^2 \underline{\tilde{d}}^{n+1}_{ABp} \cdot (\underline{\Theta}_A - \underline{\Theta}_B)} \tag{2.319}$$

From which the constraint force may be calculated. Iteration is necessary as in SHAKE.

In the second stage of QSHAKE we need to calculate another constaint force $\underline{H}_{AB}^{n+1}$ to preserve the orthogonality of the constraint bond vector and the relative velocity of the two atoms in the bond. Once again the contraint force implies corrections to the translational and rotational equations of motion, which, following the methods used above, we write directly as:

$$\begin{aligned} \underline{V}_A^{n+1} &= \tilde{\underline{V}}_A^{n+1} + \frac{\Delta t}{2M_A}\underline{H}_{AB}^{n+1} \\ \underline{\omega}_A^{n+1} &= \tilde{\underline{\omega}}_A^{n} + \frac{\Delta t}{2}h_{AB}^{n+1}\underline{U}_A^{n+1} \end{aligned} \tag{2.320}$$

where h_{AB}^{n+1} is a scalar related to the constraint force via

$$\underline{H}_{AB}^{n+1} = h_{AB}^{n+1}\underline{d}_{ABp}^{n+1}$$

Now, the velocity of the linked atom on molecule A is:

$$\underline{v}_{Ap}^{n+1} = \underline{V}_A^{n+1} + \underline{\omega}_A^{n+1} \times \underline{d}_{Ap}^{n+1} \tag{2.321}$$

which on substitution of the above equations gives:

$$\underline{v}_{Ap}^{n+1} = \tilde{\underline{v}}_{Ap}^{n+1} + \frac{\Delta t}{2}h_{AB}^{n+1}\underline{\Omega}_A^{n+1} \tag{2.322}$$

where

$$\underline{\Omega}_A^{n+1} = \left(\frac{\underline{d}_{ABp}^{n+1}}{M_A} + \underline{U}_A^{n+1}\right). \tag{2.323}$$

The constraint condition requires that

$$\underline{d}_{ABp}^{n+1} \cdot (\underline{v}_{Ap}^{n+1} - \underline{v}_{Bp}^{n+1}) = 0 \tag{2.324}$$

and substitution of the equation for $\underline{v}_{Ap}^{n+1}$ (and the equivalent for $\underline{v}_{Bp}^{n+1}$) leads directly to

$$h_{AB}^{n+1} = -\frac{2d_{ABp}^{n+1} \cdot (\tilde{\underline{v}}_{Ap}^{n+1} - \tilde{\underline{v}}_{Bp}^{n+1})}{\Delta t \underline{d}_{ABp}^{n+1} \cdot (\underline{\Omega}_A - \underline{\Omega}_B)} \tag{2.325}$$

which provides the correction for second constraint. This again requires iteration.

The VV QSHAKE algorithm is implemented in DL_POLY Classic in subroutine NVEQVV_2 with the QSHAKE constraint forces calculated in QRATTLE_R and QRATTLE_V. Again it is straightforward to couple these systems to a Hoover or Berendsen thermostat and/or barostat. The Hoover and Berendsen thermostated versions are found in NVTQVV_H2 and NVTQVV_B2 respectively. The isotropic constant pressure implementations are found in NPTQVV_H2 and NPTQVV_B2, while the anisotropic constant pressure routines are found in NSTQVV_H2 and NSTQVV_B2. The Hoover versions make use of the thermostat and barostat routines NVTQSCL, NPTQSCL_T, NPTQSCL_P, NSTQSCL_T and NSTQSCL_P according to the ensemble.

The LF QSHAKE algorithm is implemented in NVEQ_2 with the QSHAKE constraint forces applied in QSHAKE. This also has different ensemble versions: Hoover or Berendsen thermostat and/or barostat. The Hoover and Berendsen thermostated versions are found in NVTQ_H2 and NVTQ_B2 respectively. The isotropic constant pressure implementations are found in NPTQ_H2 and NPTQ_B2, while the anisotropic constant pressure routines are found in NSTQ_H2 and NSTQ_B2.

An outline of the parallel version of QSHAKE is given in section 2.6.9.

2.5.8 The DL_POLY Classic Multiple Timestep Algorithm

For simulations employing a large spherical cutoff r_{cut} radius in the calculation of the interactions DL_POLY Classic offers the possibility of using a multiple timestep algorithm to improve the efficiency. The method is based on that described by Streett *et al* [54, 55] with extension to Coulombic systems by Forester *et al* [56].

In the multiple timestep algorithm there are two cutoffs for the pair interactions: a relatively large cutoff (r_{cut}) which is used to define the standard Verlet neighbour list; and a smaller cutoff r_{prim} which is used to define a *primary list* within the larger cutoff sphere (see figure). Forces derived from atoms in the primary list are generally much larger than those derived from remaining (so-called *secondary*) atoms in the neighbour list. Good energy conservation is therefore possible if the forces derived from the primary atoms are calculated *every* timstep, while those from the secondary atoms are calculated much less frequently, and are merely extrapolated over the interval. DL_POLY Classic handles this procedure as follows.

DL_POLY Classic updates the Verlet neighbour list at irregular intervals, determined by the movement of atoms in the neighbour list (see section 2.1). The interval between updates is usually of the order of ∼20 timesteps. Partitioning the Verlet list into primary and secondary atoms always occurs when the Verlet list is updated, and thereafter at intervals of `multt` timesteps (i.e. the multi-step interval specified by the user - see section 4.1.1). Immediately after the partitioning, the force contributions from both the primary and secondary atoms are calculated. The forces are again calculated *in total* in the subsequent timestep. Thereafter, for `multt`-2 timesteps, the forces derived from the primary atoms are calculated explicitly, while those derived from the secondary atoms are calculated by linear extrapolation of the exact forces obtained in the first two timesteps of the multi-step interval. It is readily apparent how this scheme can lead to a significant saving in execution time.

Extension of this basic idea to simulations using the Ewald sum requires the following:

1. the reciprocal space terms are calculated only for the first two timesteps of the multi-step,
2. the contribution to the reciprocal space terms arising from primary interactions are immediately subtracted, leaving only the long-range components. (This is done in real space, by subtracting *erf* terms.);
3. the real space Coulombic forces arising from the secondary atoms are calculated in the first two timesteps of the multi-step using the normal Ewald expressions (i.e. the *erfc* terms).
4. the Coulombic forces arising from primary atoms are calculated at every timestep in real space assuming the *full Coulombic force*;

In this way the Coulombic forces can be handled by the same multiple timestep scheme as the van der Waals forces. The algorithm is described in detail in [56].

Note that the accuracy of the algorithm is a function of the multi-step interval `multt`, and decreases as `multt` increases. Also, the algorithm is *not time reversible* and is therefore susceptible to energy drift. Its use with a thermostat is therefore advised.

2.6 DL_POLY Parallelisation

DL_POLY Classic is a distributed parallel molecular dynamics package based on the Replicated Data parallelisation strategy [57, 58]. In this section we briefly outline the basic methodology. Users wishing to add new features DL_POLY Classic will need to be familiar with the underlying techniques as they are described (in greater detail) in references [45, 58]).

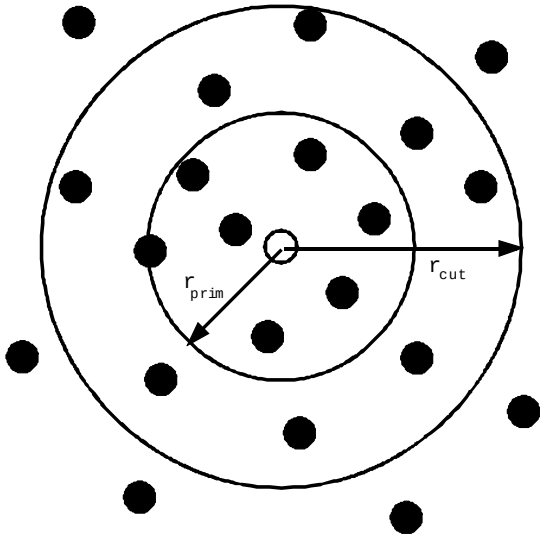

Figure 2.8: The multiple timestep algorithm

The atoms surrounding the central atom (open circle) are classified as primary if they occur within a radius r_{prim} and secondary if outside this radius but within r_{cut}. Interactions arising from primary atoms are evaluated every timestep. Interactions from secondary atoms are calculated exactly for the first two steps of a multi-step and by extrapolation afterwards.

2.6.1 The Replicated Data Strategy

The Replicated Data (RD) strategy [57] is one of several ways to achieve parallelisation in MD. Its name derives from the replication of the configuration data on each node of a parallel computer (i.e. the arrays defining the atomic coordinates $\underline{r}_i$, velocities $\underline{v}_i$ and forces $\underline{f}_i$, for all N atoms $\{i : i = 1, \ldots, N\}$ in the simulated system, are reproduced on every processing node). In this strategy most of the forces computation and integration of the equations of motion can be shared easily and equally between nodes and to a large extent be processed independently on each node. The method is relatively simple to program and is reasonably efficient. Moreover, it can be "collapsed" to run on a single processor very easily. However the strategy can be expensive in memory and have high communication overheads, but overall it has proven to be successful over a wide range of applications. These issues are explored in more detail in [57, 58].

Systems containing complex molecules present several difficulties. They often contain ionic species, which usually require Ewald summation methods [12, 59], and *intra*-molecular interactions in addition to *inter*-molecular forces. These are handled easily in the RD strategy, though the SHAKE algorithm [13] requires significant modification [45].

The RD strategy is applied to complex molecular systems as follows:

1. Using the known atomic coordinates $\underline{r}_i$, each node calculates a subset of the forces acting between the atoms. These are usually comprised of:
   (a) atom-atom pair forces (e.g. Lennard Jones, Coulombic etc.);
   (b) non-rigid atom-atom bonds;
   (c) valence angle forces;
   (d) dihedral angle forces;
   (e) improper dihedral angle forces.

2. The computed forces are accumulated in (incomplete) atomic force arrays $\underline{f}_i$ independently on each node;

3. The atomic force arrays are summed globally over all nodes;

4. The complete force arrays are used to update the atomic velocities and positions.

It is important to note that load balancing (i.e. equal and concurrent use of all processors) is an essential requirement of the overall algorithm. In DL_POLY Classic this is accomplished for the pair forces with an adaptation of the Brode-Ahlrichs scheme [23].

2.6.2 Distributing the Intramolecular Bonded Terms

DL_POLY Classic handles the intramolecular in which the atoms involved in any given bond term are explicitly listed. Distribution of the forces calculations is accomplished by the following scheme:

1. Every atom in the simulated system is assigned a unique index number from 1 to N;

2. Every intramolecular bonded term U_{type} in the system has a unique index number i_{type}: from 1 to N_{type} where *type* represents a bond, angle or dihedral.

3. A pointer array $key_{type}(n_{type}, i_{type})$ carries the indices of the specific atoms involved in the potential term labelled i_{type}. The dimension n_{type} will be 2, 3 or 4, if the term represents a bond, angle or dihedral.

4. The array $key_{type}(n_{type}, i_{type})$ is used to identify the atoms in a bonded term and the appropriate form of interaction and thus to calculate the energy and forces. Each processor is assigned the independent task of evaluating a block of $(Int(N_{total}/N_{nodes}))$ interactions.

The same scheme works for all types of bonded interactions. The global summation of the force arrays does not occur until all the force contributions, including nonbonded forces has been completed.

2.6.3 Distributing the Nonbonded Terms

In DL_POLY Classic the nonbonded interactions are handled with a Verlet neighbour list [12] which is reconstructed at intervals during the simulation. This list records the indices of all 'secondary' atoms within a certain radius of each 'primary' atom; the radius being the cut-off radius (r_{cut}) normally applied to the nonbonded potential function, plus an additional increment (Δr_{cut}). The larger radius ($r_{cut} + \Delta r_{cut}$) permits the same list to be used for several timesteps without requiring an update. The frequency at which the list must be updated clearly depends on the thickness of the region Δr_{cut}. In RD, the neighbour list is constructed *simultaneously* on each node and in such a way as to share the total burden of the work equally between nodes. Each node is responsible for a unique set of nonbonded interactions and the neighbour list is therefore different on each node. DL_POLY Classic uses a method based on the Brode-Ahlrichs scheme [23] (see figure 2.9) to construct the neighbour list.

Additional modifications are necessary to handle the excluded atoms [58]. A distributed *excluded atoms list* is constructed by DL_POLY Classic at the start of the simulation. The list is constructed so that the excluded atoms are referenced in the same order as they would appear in the Verlet neighbour list if the bonded interactions were ignored, allowing for the distributed structure of the neighbour list.

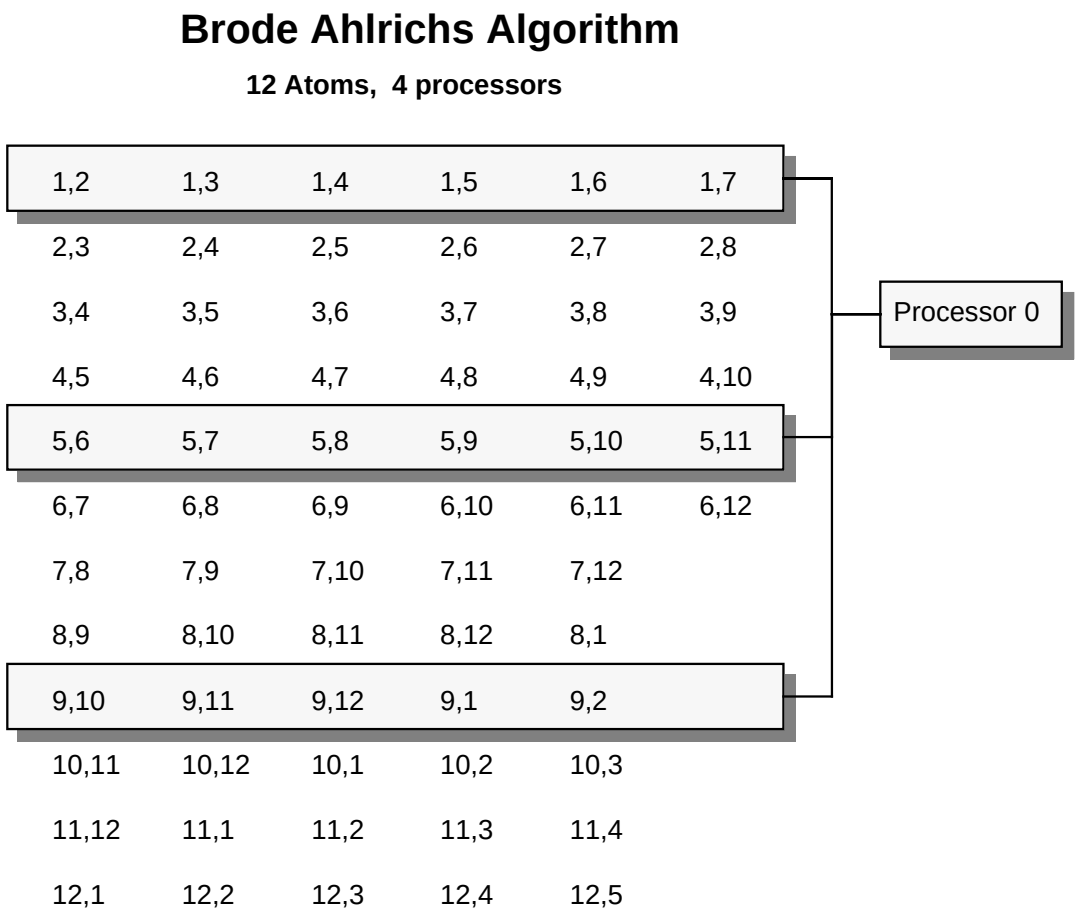

Figure 2.9: The parallel implementation of the Brode-Ahlrichs algorithm.

This diagram illustrates the reordering of the upper triangular matrix of n(n-1)/2 pair interactions so that the rows of the matrix are of approximately equally length. Each entry in the table consists of a primary atom index (constant within a row) and a "neighbouring" atom index. Rows are assigned sequentially to nodes. In the diagram node 0 deals with rows 1, 5 and 9, node 1 to rows 2, 6, and 10 etc.

When a charge group scheme (as opposed to an atomistic scheme) is used for the non-bonded terms, the group-group interactions are distributed using the Brode-Ahlrichs approach. This makes the Verlet list considerably smaller, thus saving memory, but also results in a more "coarse grain" parallelism. The consequence of which is that performance with a large number of processors will degrade more quickly than with the atomistic scheme.

Once the neighbour list has been constructed, each node of the parallel computer may proceed independently to calculate the pair force contributions to the atomic forces.

2.6.4 Modifications for the Ewald Sum

For systems with periodic boundary conditions DL_POLY Classic employs the Ewald Sum to calculate the Coulombic interactions (see section 2.4.6).

Calculation of the real space component in DL_POLY Classic employs the algorithm for the calculation of the nonbonded interactions outlined above. The reciprocal space component is cal-

culated using the schemes described in [59], in which the calculation can be parallelised by distribution of either $\underline{k}$ vectors or atomic sites. Distribution over atomic sites requires the use of a global summation of the $q_i \exp(-i\underline{k} \cdot \underline{r}_j)$ terms, but is more efficient in memory usage. Both strategies are computationally straightforward. Subroutine EWALD1 distributes over atomic sites and is often the more efficient of the two approaches. Subroutine EWALD1A distributes over the $\underline{k}$ vectors and may be more efficient on machines with large communication latencies.

Other routines required to calculate the ewald sum include EWALD2, EWALD3 and EWALD4. The first of these calculates the real space contribution, the second the self interaction corrections, and the third is required for the multiple timestep option.

2.6.5 Modifications for SPME

The SPME method requires relatively little modification for parallel computing. The real space terms are calculated exactly as they are for the normal Ewald sum, as described above. The reciprocal space sum requires a 3D Fast Fourier Transform (FFT), which in principle should be distributed over the processors, but in DL_POLY Classic the decision was made to implement a complete 3D FFT on every processor. This is expensive in memory, and potentially expensive in computer time. However a multi-processor FFT requires communication between processors and this has significant impact on the famed efficiency of the FFT. It transpires that a single processor FFT is so efficient that the adopted strategy is still effective. The charge array that is central to the SPME method (see section 2.4.7) is however built in a distributed manner and then globally summed prior to the FFT operation.

2.6.6 Three and Four Body Forces

DL_POLY Classic can calculate three/four body interactions of the valence angle type [60]. These are not dealt with in the same way as the normal nonbonded interactions. They are generally very short ranged and are most effectively calculated using a link-cell scheme [24]. No reference is made to the Verlet neighbour list nor the excluded atoms list. It follows that atoms involved in the same three/four-body term can interact via nonbonded (pair) forces and ionic forces also. The calculation of the three/four-body terms is distributed over processors on the basis of the identity of the central atom in the bond. A global summation is required to specify the atomic forces fully.

2.6.7 Metal Potentials

The simulation of metals by DL_POLY Classic makes use of density dependent potentials of the Sutton-Chen type [38]. The dependence on the atomic density presents no difficulty however, as this class of potentials can be resolved into pair contributions. This permits the use of the distributed Verlet neighbour list outlined above.

2.6.8 Summing the Atomic Forces

The final stage in the RD strategy, is the global summation of the atomic force arrays. This must be done After all the contributions to the atomic forces have been calculated. To do this DL_POLY Classic employs a global summation algorithm [57], which is generally a system specific utility.

Similarly, the total configuration energy and virial must be obtained as a global sum of the contributing terms calculated on all nodes.

2.6.9 The SHAKE, RATTLE and Parallel QSHAKE Algorithms

The SHAKE and RATTLE algorithms are methods for constraining rigid bonds. Parallel adaptations of both are couched in the Replicated Data strategy. The essentials of the methods are as follows.

1. The bond constraints acting in the simulated system are shared equally between the processing nodes.
2. Each node makes a list recording which atoms are bonded by constraints it is to process. Entries are zero if the atom is not bonded.
3. A copy of the array is passed to each other node in turn. The receiving node compares the incoming list with its own and keeps a record of the shared atoms and the nodes which share them.
4. In the first stage of the SHAKE algorithm, the atoms are updated through the usual Verlet algorithm, without regard to the bond constraints.
5. In the second (iterative) stage of SHAKE, each node calculates the incremental correction vectors for the bonded atoms in its own list of bond constraints. It then sends specific correction vectors to all neighbours that share the same atoms, using the information compiled in step 3.
6. When all necessary correction vectors have been received and added the positions of the constrained atoms are corrected.
7. Steps 5 and 6 are repeated until the bond constraints are converged.
8. After convergence the coordinate arrays on each node are passed to all the other nodes. The coordinates of atoms that are *not* in the constraint list of a given node are taken from the incoming arrays (an operation we term *splicing*).
9. Finally, the change in the atom positions is used to calculate the atomic velocities.

The above scheme is complete for a implementation based on the leapfrog integration algorithm. However a velocity Verlet (VV) scheme requires additional steps.

1. Step 9 above does not apply for VV. The velocity is integrated under the normal VV scheme.
2. When the velocity is updated, iteration of the constraint force takes place. The incremental changes to the velocity are communicated between nodes sharing constrained atoms as for the bondlength constraints.
3. Iteration is repeated until the bond constraints are converged.
4. After convergence the velocity arrays on each node are passed to all the other nodes by *splicing*.

This scheme contains a number of non-trivial operations, which are described in detail in [45]. However some general comments are worth making.

The compilation of the list of constrained atoms on each node, and the circulation of the list (items 1 - 3 above) need only be done once in any given simulation. It also transpires that in sharing bond contraints between nodes, there is an advantage to keeping as many of the constraints

pertaining to a particular molecule together on one node as is possible within the requirement for load balancing. This reduces the data that need to be transferred between nodes during the iteration cycle. It is also advantageous, if the molecules are small, to adjust the load balancing between processors to prevent shared atoms. The loss of balance is compensated by the elimination of communications during the SHAKE cycle. These techniques are exploited by DL_POLY Classic.

The QSHAKE algorithm is an extension of the SHAKE algorithm for constraint bonds between rigid bodies. The parallel strategy is very similar to that of SHAKE. The only significant difference is that increments to the atomic forces, not the atomic positions, are passed between processors at the end of each iteration.

Chapter 3

Construction and Execution

Scope of Chapter

This chapter describes how to compile a working version of DL_POLY Classic and how to run it.

3.1 Constructing DL_POLY Classic

3.1.1 Overview

The DL_POLY Classic executable program is constructed as follows.

1. DL_POLY Classic is supplied as a UNIX compressed tar file. This must uncompressed and un-tared to create the DL_POLY Classic directory (section 1.4).

2. In the *build* subdirectory you will find the required DL_POLY Classic makefile (see section 3.2.1 and Appendix A, where a sample Makefile is listed). This must be copied into the subdirectory containing the relevant source code. In most cases this will be the *source* subdirectory.

3. The makefile is executed with the appropriate keywords (section 3.2.1) which selects for specific computers (including serail and parallel machines) and the appropriate communication software.

4. The makefile produces the executable version of the code, which as a default will be named DLPOLY.X and located in the *execute* subdirectory.

5. DL_POLY also has a Java GUI. The files for this are stored in the subdirectory *java*. Compilation of this is simple and requires running the javac compiler and the jar utility. Details for these procedures are provided in the GUI manual [9].

6. To run the executable for the first time you require the files CONTROL, FIELD and CONFIG (and possibly TABLE or TABEAM if you have tabulated potentials). These must be present in the directory from which the program is executed. (See section 4.1 for the description of the input files.)

7. Executing the program will produce the files OUTPUT, REVCON and REVIVE (and optionally STATIS, HISTORY, RDFDAT and ZDNDAT) in the executing directory. (See section 4.2 for the description of the output files.)

This simple procedure is enough to create a standard version to run most DL_POLY Classic applications. However it sometimes happens that additional modifications may be necessary.

On starting, DL_POLY Classic scans the input data and makes an estimate of the sizes of the arrays it requires to do the simulation. Sometimes the estimates are not good enough. The most common occurrences of this are NPT and NST simulations, or simulations where the local density on the MD cell may significantly exceed the mean density of the cell (systems with a vacuum gap for example). Under these circumstances arrays initally allocated may be insufficent. In which case DL_POLY Classic may report a memory problem and request that you recompile the code with hand-adjusted array dimensions. This topic is dealt with more fully in Appendix C.

3.2 Compiling and Running DL_POLY Classic

3.2.1 Compiling the Source Code

When you have obtained DL_POLY Classic from Daresbury Laboratory and unpacked it, your next task will be to compile it. To aid compilation a set of makefiles has been provided in the sub-directory *build* (see example in Appendix A of this document). The versions go by the names of:

- MakePAR - to build a parallel MPI version on a unix platform;
- MakeSEQ - to build a sequential (one processor) unix version;
- MakeWIN - to build a Windows (one processor, XP) version.

Select the one you need and copy it into the *source* directory. (In what follows we assume the makefile in the *source* directory is called 'Makefile'.) The Makefile will build an executable with a wide range of functionality - sufficient for the test cases and for most users' requirements.

Users will need to modify the Makefile if they are to add additional functionality to the code, or if it requires adaptation for a non specified computer. Modifications may also be needed for the Smoothed Particle Mesh Ewald method if a system specific 3D FFT routine is desired (see below: "Modifying the makefile").

Note the following system requirements for a successful build of DL_POLY Classic.

1. A FORTRAN 90 compiler;
2. The Java SDK from Sun Microsystems (if the GUI is required).
3. A UNIX operating system (or Windows XP with CygWin, if a PC version is required).

Run the Makefile you copied from the *build* sub-directory in the *source* sub-directory. It will create the executable in the *execute* sub-directory. The compilation of the program is initiated by typing the command:

make target

where *target* is the specification of the required machine (e.g. hpcx). For many computer systems this is all that is required to compile a working version of DL_POLY Classic. (To determine which targets are already defined in the makefile, typing the command *make* without a nominated target will produce a list of known targets.)

The full specification of the *make* command is as follows

make <TARGET= . . . > < EX=. . . > < BINROOT=. . . >

where some (or all) of the keywords may be omitted. The keywords and their uses are described below. Note that keywords may also be set in the unix environment (e.g. with the "setenv" command in a C-shell).

For PCs running Windows, the makefile assumes the user has installed the Cygwin Unix API available from

http : //sources.redhat.com/cygwin

The recommended FORTRAN 90 compiler is GFORTRAN, but G95 can also be used, see:

http : //ftp.g95.org/

.

3.2.1.1 Keywords for the Makefile

1. **TARGET**

   The TARGET keyword indicates which kind of computer the code is to be compiled for.

This **must** be specifed - there is no default value. Valid targets can be listed by the makefile if the command *make* is typed, without arguments. The list frequently changes as more targets are added and redundant ones removed. Users are encouraged to extend the Makefile for themselves, using existing targets as examples.

2. **EX**

   The EX keyword specifies the executable name. The default name for the executable is "DLPOLY.X".

3. **BINROOT**

   The BINROOT keyword specifies the directory in which the executable is to be stored. The default setting is "../execute".

3.2.1.2 Modifying the Makefile

1. **Changing the TARGET**

   If you do not intend to run DL_POLY Classic on one of the specified machines, you must add appropriate lines to the makefile to suit your circumstances. The safest way to do this is to modify an existing TARGET option for your purposes. The makefile supplied with DL_POLY Classic contains examples for different serial and parallel (MPI) environments, so you should find one close to your requirements. You must of course be familiar with the appropriate invocation of the FORTRAN 90 compiler for your local machine and also any alternatives to MPI your local machine may be running. If you wish to compile for MPI systems remember to ensure the appropriate library directories are accessible to you. If you require a serial version of the code, you must remove references to the MPI libraries from the Makefile and add the file serial.f to your compilation - this will insert replacement (dummy) routines for the MPI calls.

2. **Enabling the Smoothed Particle Mesh Ewald**

   The standard compilation of DL_POLY Classic will incorporate a basic 3D Fast Fourier Transform (FFT) routine to enable the SPME functionality. Users may wish to try alternative FFT routines, which may offer faster performance. Some 'hooks' for these appear in the code as comment lines in the FORTRAN source. The user should search for the following keys in the code:

   - CCRAY - for the Cray FFT routines;
   - CFFTW - for the FFTW public domain FFT routines;
   - CESSL - for the IBM scientific library FFT routines;
   - CSGIC - for the Silicon Graphics FFT routines.

   The appropriate lines should be uncommented and the references to the DLPFFT3 subroutine should be commented out before compiling.

3. **Problems with optimization ?**

Some subroutines may not compile correctly when using optimization on some compilers. This is not necessarily the fault of the DL_POLY Classic code, some compilers are just flakey. This can be circumvented by compiling the offending subroutines separately with optimisation flags turned off.

4. **Adding new functionality**

   To include a new subroutine in the code simply add *subroutine*.o to the list of object names in the makefile. The simplest way is to add names to the "OBJ_SRC" list. However, for more substantial modifications it is advisable to construct a proper F90 module containing several related subroutines and add this to the "OBJ_MOD" list.

3.2.1.3 Note on Interpolation

In DL_POLY Classic the short-range (Van der Waals) contributions to energy and force are evaluated by interpolation of tables constructed at the beginning of execution. DL_POLY Classic employs a 3-point interpolation scheme.

A guide to the *minimum* number of grid points (`mxgrid`) required for interpolation in r to give good energy conservation in a simulation is:

$$\texttt{mxgrid} \geq 100(\texttt{rcut}/\texttt{rmin})$$

where `rmin` is the *smallest* position minimum of the non-bonded potentials in the system. The parameter `mxgrid` is defined in the DL_PARAMS.INC file, and must be set before compilation.

A utility program TABCHK is provided in the DL_POLY *utility* sub-directory to help users choose a sufficiently accurate interpolation scheme (including array sizes) for their needs.

3.2.2 Running DL_POLY Classic

To run the DL_POLY Classic executable (DLPOLY.X), for most applications, you will initially require three, possibly four, input data files, which you must create in the *execute* sub-directory, (or whichever sub-directory you keep the executable program.) The first of these is the CONTROL file (section 4.1.1), which indicates to DL_POLY Classic what kind of simulation you want to run, how much data you want to gather and for how long you want the job to run. The second file you need is the CONFIG file (section 4.1.2). This contains the atom positions and, depending on how the file was created (e.g. whether this is a configuration created from 'scratch' or the end point of another run), the velocities also. The third file required is the FIELD file (section 4.1.3), which specifies the nature of the intermolecular interactions, the molecular topology and the atomic properties, such as charge and mass. Sometimes you will also require a TABLE file (section 4.1.5), which contains the potential and force arrays for functional forms not available within DL_POLY Classic (usually because they are too complex e.g. spline potentials). Sometimes you will also require a TABEAM file (section 4.1.6), if your simulation includes embedded atom potentials for metallic systems.

Examples of input files are found in the *data* sub-directory, which can be copied into the *execute* subdirectory using the *select* macro found in the *execute* sub-directory.

A successful run of DL_POLY Classic will generate several data files, which appear in the *execute* sub-directory. The most obvious one is the file OUTPUT (section 4.2.2), which provides an effective summary of the job run: the input information; starting configuration; instantaneous and rolling-averaged thermodynamic data; final configurations; radial distribution functions (RDFs); and job timing data. The OUTPUT file is human readable. Also present will be the restart files REVIVE (section 4.2.5) and REVCON (section 4.2.3). REVIVE contains the accumulated data for

a number of thermodynamic quantities and RDFs, and is intended to be used as the input file for a following run. It is *not* human readable. The REVCON file contains the *restart configuration* i.e. the final positions, velocities and forces of the atoms when the run ended and is human readable. The STATIS file (section 4.2.8) contains a catalogue of instantaneous values of thermodynamic and other variables, in a form suitable for temporal or statistical analysis. Finally, the HISTORY file (section 4.2.1) provides a time ordered sequence of configurations to facilitate further analysis of the atomic motions. Depending on which version of the TRAJECT subroutine you compiled in the code, this file may be either formatted (human readable) or unformatted. You may move these output files back into the *data* sub-directory using the *store* macro found in the *execute* sub-directory.

Note that versions of DL_POLY Classic after 2.10 may also create the files RDFDAT and ZDNDAT, containing the RDF and Z-density data respectively. They are both human readable files.

3.2.3 Restarting DL_POLY Classic

The best approach to running DL_POLY Classic is to define from the outset precisely the simulation you wish to perform and create the input files specific to this requirement. The program will then perform the requested simulation, but may terminate prematurely through error, inadequate time allocation or computer failure. Errors in input data are your responsibility, but DL_POLY Classic will usually give diagnostic messages to help you sort out the trouble. Running out of job time is common and provided you have correctly specified the job time variables (using the **close time** and **job time** directives - see section 4.1.1) in the CONTROL file, DL_POLY Classic will stop in a controlled manner, allowing you to restart the job as if it had not been interrupted.

To restart a simulation after normal termination you will again require the CONTROL file, the FIELD (and TABLE) file, and a CONFIG file, which is the exact copy of the REVCON file created by the previous job. You will also require a new file: REVOLD (section 4.1.4), which is an exact copy of the previous REVIVE file. If you attempt to restart DL_POLY Classic without this additional file available, the job will fail. Note that DL_POLY Classic will append new data to the existing STATIS and HISTORY files if the run is restarted, other output files will be **overwritten**.

In the event of machine failure, you should be able to restart the job in the same way from the surviving REVCON and REVIVE files, which are dumped at intervals to meet just such an emergency. In this case check carefully that the input files are intact and use the HISTORY and STATIS files with caution - there may be duplicated or missing records. The reprieve processing capabilities of DL_POLY Classic are not foolproof - the job may crash while these files are being written for example, but they can help a great deal. You are advised to keep backup copies of these files, noting the times they were written, to help you avoid going right back to the start of a simulation.

You can also extend a simulation beyond its initial allocation of timesteps, provided you still have the REVCON and REVIVE files. These should be copied to the CONFIG and REVOLD files respectively and the directive **timesteps** adjusted in the CONTROL file to the new total number of steps required for the simulation. For example if you wish to extend a 10000 step simulation by a further 5000 steps use the directive **timesteps 15000** in the CONTROL file and include the **restart** directive.

Note that you can use the **restart scale** directive if you want to reset the temperature at the restart, but note also that this also resets all internal accumulators (timestep included) to zero. Alternatively you can use the **restart noscale** directive if you want to leave the atomic velocities unchanged at restart, but wish to start a fresh simulation. This will also reset internal accumulators and timestep number to zero. Both the **restart scale** and **restart noscale** options will therefore ignore the REVOLD file.

3.2.4 Optimising the Starting Structure

The preparation of the initial structure of a system for a molecular dynamics simulation can be difficult. It is quite likely that the structure created does not correspond to one typical of the equilibrium state for the required state point, for the given force field employed. This can make the simulation unstable in the initial stages and can even prevent it from proceeding.

For this reason DL_POLY Classic has available a selection of structure relaxation methods. Broadly speaking, these are energy minimisation algorithms, but their role in DL_POLY Classic is not to provide users with true structural optimisation procedures capable of finding the ground state structure They are simply intended to help users improve the quality of the starting structure prior to a dynamical simulation.

The available algorithms are:

1. 'Zero' temperature molecular dynamics . This is equivalent to a dynamical simulation at low temperature. At each time step the molecules move in the direction of the computed forces (and torques), but are not allowed to acquire a velocity larger than that corresponding to a temperature of 1 Kelvin. The subroutine that performs this procedure is ZERO_KELVIN, which is found in the file OPTIMISER_MODULE.F.

2. Conjugate Gradients (CG) minimisation . This is nominally a simple minimisation of the system configuration energy using the conjugate gradients method [61]. The algorithm coded into DL_POLY Classic allows is an adaptation that allows for rotation and translation of rigid bodies. Rigid (contraint) bonds however are treated as stiff harmonic springs - a strategy which we find does allow the bonds to converge within the accuracy required by SHAKE. The subroutine that performs this procedure is STRUCOPT, which is found in the file OPTIMISER_MODULE.F.

3. 'Programmed' energy minimisation, involving both molecular dynamics and conjugate gradients . This method combines conjugate gradient minimisation with molecular dynamics. Minimisation is followed by user-defined intervals of (usually low temperature) dynamics, in a cycle of minimisation - dynamics - minimisation etc, which is intended to help the structure relax from overstrained conditions. When using the programmed minimisation DL_POLY Classic writes (and rewrites) the file CFGMIN 4.2.4, which represents the lowest energy structure found during the programmed minimisation. CFGMIN is written in CONFIG file format (see section 4.1.2) and can be used in place of the original CONFIG file.

It should be noted that none of these algorithms permit the simulation cell to change shape. It is only the atomic structure that is relaxed. After which it is assumed that normal molecular dynamics will commence from the final structure.

Comments on the Minimisation Procedures

1. The zero temperature dynamics is really dynamics conducted at 1 Kelvin. However the dynamics has been modified so that the velocities of the atoms are always directed along the force vectors. Thus the dynamics follows the steepest descent to the (local) minimum. From any given configuration, it will always descend to the same minimum.

2. The conjugate gradient procedure has been adapted to take account of the possibilites of constraint bonds and rigid bodies being present in the system. If neither of these is present, the conventional unadapted procedure is followed.

(a) In the case of rigid bodies, atomic forces are resolved into molecular forces and torques. The torques are subsequently transformed into an equivalent set of atomic forces which are perpendicular both to the instantaneous axis of rotation (defined by the torque vector) and to the cylindrical radial displacement vector of the atom from the axis. These modified forces are then used in place of the original atomic forces in the conjugate gradient scheme. The atomic displacement induced in the conjugate gradient algorithm is corrected to maintain the magnitude of the radial position vector, as required for circular motion.

(b) With regard to constraint bonds, these are replaced by stiff harmonic bonds to permit minimisation. This is not normally recommended as a means to incorporate constraints in minimisation procedures as it leads to ill conditioning. However, *if the constraints in the original structure are satisfied*, we find that provided only small atomic displacements are allowed during relaxation it is possible to converge to a minimum energy structure. Furthermore, provided the harmonic springs are stiff enough, it is possible afterwards to satisfy the constraints exactly by further optimising the structure using the stiff springs alone, without having a significant affect on the overall system energy.

(c) Systems with independent constraint bonds and rigid bodies and systems with rigid bodies linked by constraints may also be minimised by these methods.

3. Of the three minimisation methods available in DL_POLY Classic, only the programmed minimiser is capable of finding more than one minimum without the user intervening.

4. Finally, we emphasise once again that the purpose of the minimisers in DL_POLY Classic is to help improve the quality of the starting structure and we believe they are adequate for that purpose. We do not recommend them as general molecular structure optimisers. They may however prove useful for relaxing crystal structures to 0 Kelvin for the purpose of identifying a true crystal structure.

3.2.5 Choosing Ewald Sum Variables

3.2.5.1 Ewald sum and SPME

This section outlines how to optimise the accuracy of the Ewald sum parameters for a given simulation. In what follows the directive **spme** may be used anywhere in place of the directive **ewald** if the user wishes to use the Smoothed Particle Mesh Ewald method.

As a guide to beginners DL_POLY Classic will calculate reasonable parameters if the **ewald precision** directive is used in the CONTROL file (see section 4.1.1). A relative error (see below) of 10^{-6} is normally sufficient so the directive

ewald precision 1d-6

will cause DL_POLY Classic to evaluate its best guess at the Ewald parameters α, `kmax1`, `kmax2` and `kmax3`. (The user should note that this represents an *estimate*, and there are sometimes circumstances where the estimate can be improved upon. This is especially the case when the system contains a strong directional anisotropy, such as a surface.) These four parameters may also be set explicitly by the **ewald sum** directive in the CONTROL file. For example the directive

ewald sum 0.35 6 6 8

would set $\alpha = 0.35$ Å^{-1}, `kmax1` = 6, `kmax2 = 6` and `kmax3` = 8. The quickest check on the accuracy of the Ewald sum is to compare the Coulombic energy (U) and the coulombic virial ($\mathcal{W}$)

in a short simulation. Adherence to the relationship $U = -\mathcal{W}$ shows the extent to which the Ewald sum is correctly converged. These variables can be found under the columns headed `eng_cou` and `vir_cou` in the OUTPUT file (see section 4.2.2).

The remainder of this section explains the meanings of these parameters and how they can be chosen. The Ewald sum can only be used in a three dimensional periodic system. There are three variables that control the accuracy: α, the Ewald convergence parameter; $r_{\rm cut}$ the real space forces cutoff; and the `kmax1,2,3` integers [1] that effectively define the range of the reciprocal space sum (one integer for each of the three axis directions). These variables are not independent, and it is usual to regard one of them as pre-determined and adjust the other two accordingly. In this treatment we assume that $r_{\rm cut}$ (defined by the **cutoff** directive in the CONTROL file) is fixed for the given system.

The Ewald sum splits the (electrostatic) sum for the infinite, periodic, system into a damped real space sum and a reciprocal space sum. The rate of convergence of both sums is governed by α. Evaluation of the real space sum is truncated at $r = r_{\rm cut}$ so it is important that α be chosen so that contributions to the real space sum are negligible for terms with $r > r_{\rm cut}$. The relative error (ϵ) in the real space sum truncated at $r_{\rm cut}$ is given approximately by

$$\epsilon \approx \mathrm{erfc}(\alpha r_{\rm cut})/r_{\rm cut} \approx \exp[-(\alpha.r_{\rm cut})^2]/r_{\rm cut} \tag{3.1}$$

The recommended value for α is $3.2/r_{\rm cut}$ or greater (too large a value will make the reciprocal space sum very slowly convergent). This gives a relative error in the energy of no greater than $\epsilon = 4 \times 10^{-5}$ in the real space sum. When using the directive **ewald precision** DL_POLY Classic makes use of a more sophisticated approximation:

$$\mathrm{erfc}(x) \approx 0.56\exp(-x^2)/x \tag{3.2}$$

to solve recursively for α, using equation 3.1 to give the first guess.

The relative error in the reciprocal space term is approximately

$$\epsilon \approx \exp(-k_{max}^2/4\alpha^2)/k_{max}^2 \tag{3.3}$$

where

$$k_{max} = \frac{2\pi}{L}\ \texttt{kmax} \tag{3.4}$$

is the largest k-vector considered in reciprocal space, L is the width of the cell in the specified direction and `kmax` is an integer.

For a relative error of 4×10^{-5} this means using $k_{max} \approx 6.2\alpha$. `kmax` is then

$$\texttt{kmax} > 3.2\ L/r_{\rm cut} \tag{3.5}$$

In a cubic system, $r_{\rm cut} = L/2$ implies `kmax` $= 7$. In practice the above equation slightly over estimates the value of `kmax` required, so optimal values need to be found experimentally. In the above example `kmax` $= 5$ or 6 would be adequate.

If your simulation cell is a truncated octahedron or a rhombic dodecahedron then the estimates for the `kmax` need to be multiplied by $2^{1/3}$. This arises because twice the normal number of k-vectors are required (half of which are redundant by symmetry) for these boundary contributions [45].

If you wish to set the Ewald parameters manually (via the **ewald sum** or *spme sum* directives) the recommended approach is as follows. Preselect the value of $r_{\rm cut}$, choose a working a value of

[1] **Important note:** For the SPME method the values of `kmax1,2,3` should be double those obtained in this prescription, since they specify the sides of a cube, not a radius of convergence.

α of about $3.2/r_{\rm cut}$ and a large value for the `kmax` (say 10 10 10 or more). Then do a series of ten or so *single* step simulations with your initial configuration and with α ranging over the value you have chosen plus and minus 20%. Plot the Coulombic energy (and $-\mathcal{W}$) versus α. If the Ewald sum is correctly converged you will see a plateau in the plot. Divergence from the plateau at small α is due to non-convergence in the real space sum. Divergence from the plateau at large α is due to non-convergence of the reciprocal space sum. Redo the series of calculations using smaller `kmax` values. The optimum values for `kmax` are the smallest values that reproduce the correct Coulombic energy (the plateau value) and virial at the value of α to be used in the simulation.

Note that one needs to specify the three integers (`kmax1, kmax2, kmax3`) referring to the three spatial directions, to ensure the reciprocal space sum is equally accurate in all directions. The values of `kmax1`, `kmax2` and `kmax3` must be commensurate with the cell geometry to ensure the same minimum wavelength is used in all directions. For a cubic cell set `kmax1` = `kmax2` = `kmax3`. However, for example, in a cell with dimensions $2A = 2B = C$ (ie. a tetragonal cell, longer in the c direction than the a and b directions) use 2`kmax1` = 2`kmax2` = (`kmax3`).

If the values for the `kmax` used are too small, the Ewald sum will produce spurious results. If values that are too large are used, the results will be correct but the calculation will consume unnecessary amounts of *cpu* time. The amount of *cpu* time increases with `kmax1` × `kmax2` × `kmax3`.

3.2.5.2 Hautman Klein Ewald Optimisation

Setting the HKE parameters can also be achieved rather simply, by the use of a **hke precision** directive in the CONTROL file e.g.

hke precision 1d-6 1 1

which specifies the required accuracy of the HKE convergence functions, plus two additional integers; the first specifying the order of the HKE expansion (`nhko`) and the second the maximum lattice parameter (`nlatt`). DL_POLY Classic will permit values of `nhko` from 1-3, meaning the HKE Taylor series expansion may range from zeroth to third order. Also `nlatt` may range from 1-2, meaning that (1) the nearest neighbour, and (2) and next nearest neighbour, cells are explicitly treated in the real space part of the Ewald sum. Increasing either of these parameters will increase the accuracy, but also substantially increase the cpu time of a simulation. The recommended value for both these parameters is 1 and if *both* these integers are left out, the default values will be adopted.

As with the standard Ewald and SPME methods, the user may set alternative control parameters with the CONTROL file **hke sum** directive e.g.

hke sum 0.05 6 6 1 1

which would set $\alpha = 0.05$ Å^{-1}, `kmax1 = 6`, `kmax2 = 6`. Once again one may check the accuracy by comparing the Coulombic energy with the virial, as described above. The last two integers specify, once again, the values of `nhko` and `nlatt` respectively. (Note it is possible to set either of these to zero in this case.)

Estimating the parameters required for a given simulation follows a similar procedure as for the standard Ewald method (above), but is complicated by the occurrence of higher orders of the convergence functions. Firstly a suitable value for α may be obtained when `nlatt`=0 from the rule: $\alpha = \beta/r_{cut}$, where r_{cut} is the largest real space cutoff compatible with a single MD cell and β=(3.46,4.37,5.01,5.55) when `nhko`=(0,1,2,3) respectively. Thus in the usual case where `nhko`=1, β=4.37. When `nlatt`$\neq$0, this β value is multiplied by a factor $1/(2 * nlatt + 1)$.

The estimation of `kmax1,2` is the same as that for the standard Ewald method above. Note

that if any of these parameters prove to be insufficiently accurate, DL_POLY Classic will issue an error in the OUTPUT file, and indicate whether it is the real or reciprocal space sums that is questionable.

3.3 DL_POLY Classic Error Processing

3.3.1 The DL_POLY Classic Internal Error Facility

DL_POLY Classic contains a number of in-built error checks scattered throughout the package which detect a wide range of possible errors. In all cases, when an error is detected the subroutine ERROR is called, resulting in an appropriate message and termination of the program execution (either immediately, or after additional processing.).

Users intending to insert new error checks should ensure that all error checks are performed *concurrently* on *all* nodes, and that in circumstances where a different result may obtain on different nodes, a call to the global status routine GSTATE is made to set the appropriate global error flag on all nodes. Only after this is done, a call to subroutine ERROR may be made. An example of such a procedure might be:

```
logical safe
safe=(test_condition)
call gstate(safe)
if(.not.safe) call error(node_id,message_number)
```

In this example it is assumed that the logical operation *test_condition* will result in the answer *.true.* if it is safe for the program to proceed, and *.false.* otherwise. The call to ERROR requires the user to state the identity of the calling node (`node_id`), so that only the nominated node in ERROR (i.e. node 0) will print the error message. The variable `message_number` is an integer used to identify the appropriate message to be printed.

In all cases, if ERROR is called with a *non-negative* message number, the program run terminates. If the message number is *negative*, execution continues, but even in this case DL_POLY Classic will terminate the job at a more appropriate place. This feature is used in processing the CONTROL and FIELD file directives. A possible modification users may consider is to dump additional data before the call to ERROR is made.

A full list of the DL_POLY Classic error messages and the appropriate user action can be found in Appendix C of this document.

Chapter 4

Data Files

Scope of Chapter

This chapter describes all the input and output files for DL_POLY Classic, examples of which are to be found in the *data* sub-directory.

4.1 The INPUT files

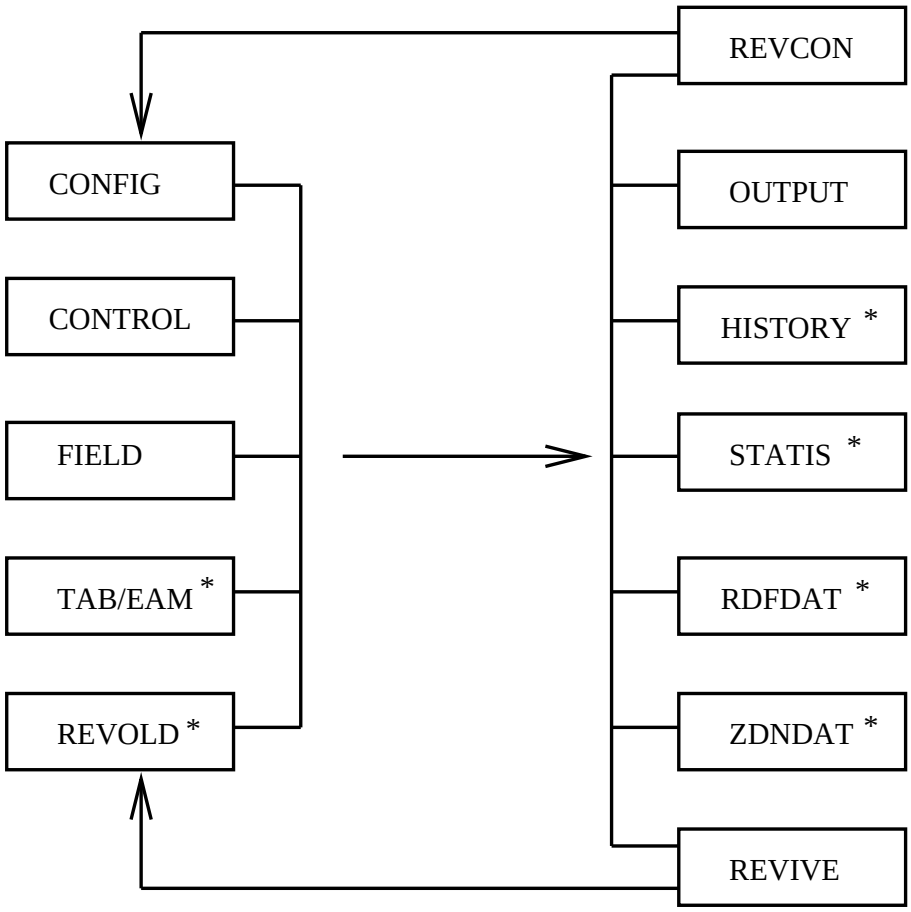

Figure 4.1: DL_POLY Classic input and output files

.

Input files appear on the left and output files on the right. Files marked with an asterisk are non-mandatory. File CFGMIN (not shown) appears as an output file if the user selects the programmed minimisation option (see 3.2.4).

In normal use DL_POLY Classic requires six input files named CONTROL, CONFIG, FIELD, TABLE, TABEAM and REVOLD. The first three files are mandatory, while files TABLE and TABEAM (TAB/EAM in the figure) are used only to input certain kinds of pair potential, and are not always required. REVOLD is required only if the job represents a continuation of a previous job. In the following sections we describe the form and content of these files.

Note: In addition to the files described in this chapter, users of the hyperdynamics features of DL_POLY Classic should see Chapter 5, where additional files specific to that purpose are described.

Users are strongly advised to study the example input files appearing in the *data* sub-directory to see how different files are constructed.

4.1.1 The CONTROL File

The CONTROL file is read by the subroutine SIMDEF and defines the control variables for running a DL_POLY Classic job. It makes extensive use of **directives** and **keywords**. Directives are

character strings that appear as the first entry on a data record (or line) and which invoke a particular operation or provide numerical parameters. Also associated with each directive may be one or more keywords, which may qualify a particular directive by, for example, adding extra options. Directives have the following general form:

keyword [*options*] {`data`}

The keyword and options are text fields, while the data options are numbers (integers or reals).

Directives can appear in any order in the CONTROL file, except for the **finish** directive which marks the end of the file. Some of the directives are mandatory (for example the **timestep** directive that defines the timestep), others are optional.

This way of constructing the file is very convenient, but it has inherent dangers. It is, for example, quite easy to specify the same directive more than once, or specify contradictory directives, or invoke algorithms that do not work together. By and large DL_POLY Classic tries to sort out these difficulties and print helpful error messages, but it does not claim to be foolproof. Fortunately in most cases the CONTROL file will be small and easy to check visually. It is important to think carefully about a simulation beforehand and ensure that DL_POLY Classic is being asked to do something that is physically reasonable. It should also be remembered that the present capabilites the package may not allow the simulation required and it may be necessary for you yourself to add new features.

An example CONTROL file appears below. The directives and keywords appearing are described in the following section.

```
DL_POLY TEST CASE 1: K Na disilicate glass

temperature      1000.0
pressure         0.0000
ensemble nve

integrator leapfrog
steps               500
equilibration       200
multiple              5
scale                10
print                10
stack               100
stats                10
rdf                  10

timestep         0.0010
primary          9.0000
cutoff           12.030
delr             1.0000
rvdw             7.6000
ewald precision  1.0E-5
print rdf

job time              1200.0
close time            100.00
```

```
finish
```

4.1.1.1 The CONTROL file format

The file is free-formatted, integers, reals and additional keywords are entered following the keyword on each record. Real and integer numbers must be separated by a non-numeric character (preferably a space or comma) to be correctly interpreted. No logical variables appear in the control file. Comment records (beginning with a #) and blank lines may be added to aid legibility (see example above). The CONTROL file is not case sensitive.

- The first record in the CONTROL file is a header 80 characters long, to aid identification of the file.
- The last record is a **finish** directive, which marks the end of the input data.

Between the header and the **finish** directive, a wide choice of control directives may be inserted. These are described below.

4.1.1.2 The CONTROL File Directives

Users of the hyperdynamics features of DL_POLY Classic (including nudged elastic band calculations) should also consult Chapter5, where additional CONTROL directives specific to this function are described. Similarly, users of the solvation features (energy decomposition, free energy and solvation induced spectral shifts) should consult Chapter 6. The directives available for other functions are as follows.

directive:	**meaning:**
all pairs	Use all pairs for calculating electrostatic interactions with multiple time step method
cap *f*	Cap forces during equilibration period *f* is maximum cap in units of kT/Å (default *f*=1000)
close time *f*	Set job closure time to *f* seconds
collect	Include equilibration data in overall statistics
coul	Calculate coulombic forces
cut *f*	Set required forces cutoff to *f* (Å)
densvar *f*	Percentage density variation for arrays
distan	Calculate coulombic forces using distance dependent dielectric
delr *f*	Set Verlet neighbour list shell width to *f* (Å)
ensemble nve	Select NVE ensemble (default)
ensemble nvt ber *f*	Select NVT ensemble with Berendsen thermostat with relaxation constant *f* (ps)
ensemble nvt evans	Select NVT ensemble with Evans thermostat
ensemble nvt hoover *f*	Select NVT ensemble with Hoover-Nose thermostat with relaxation constant *f* (ps)
ensemble npt ber f_1 f_2	

Select Berendsen NPT ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble npt hoover f_1 f_2
Select Hoover NPT ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble nst ber f_1 f_2
Select Berendsen N$\underline{\underline{\sigma}}$T ensemble, with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble nst hoover f_1 f_2
Select Hoover N$\underline{\underline{\sigma}}$T ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble pmf Select (NVE) potential of mean force ensemble

eps f Set relative dielectric constant to f (default 1.0)

equil n Equilibrate simulation for first n timesteps

ewald precision f Select Ewald sum for electrostatics, with
automatic parameter optimisation ($0 < f < 1.E-4$)

ewald sum α *k1 k2 k3*
Select Ewald sum for electrostatics, with:
α = Ewald convergence parameter (Å^{-1})
k1 = maximum k-vector index in x-direction
k2 = maximum k-vector index in y-direction
k3 = maximum k-vector index in z-direction

finish Close the CONTROL file (last data record)

hke precision f i j Select HK-Ewald sum for electrostatics, with
automatic parameter optimisation ($0 < f < .5$)
i = required order of HKE expansion (recommend 1)
j = required lattice sum order (recommend 1)

hke sum α *k1 k2 i j*
Select HK-Ewald sum for electrostatics, with:
α = Ewald convergence parameter (Å^{-1})
k1 = maximum g-vector index in x-direction
k2 = maximum g-vector index in y-direction
nhko = required order of HKE expansion (recommend 1)
nlatt = required lattice sum order (recommend 1)

integrator *type* Select *type* of integration algorithm:
leapfrog : leapfrog integration algorithm (default)
velocity : velocity Verlet integration algorithm.
The default is *leapfrog* if **integrator** is not specified.

impact i n E ux uy uz
Select impact dynamics with:
i = identity of impacted atom
n = time step when impact occurs
E = the recoil energy of the impacted atom (in KeV)
ux = X-component of normalised recoil direction vector
uy = Y-component of normalised recoil direction vector
uz = Z-component of normalised recoil direction vector

job time f Set job time to f seconds

minim energy n f Programmed minimisation based on energy, force or position, with:

minim force n f n = number of time steps between minimisations, and

minim position *n f*	*f* = permitted variation (tolerance) (DL_POLY units).
mult *n*	Set multiple timestep (multi-step)interval (activated when *n*>2)
no elec	Ignore all coulombic interactions
no fic	Activate 'flying ice cube' prevention for Berendsen NVT, NPT etc.
no link	Do not use link cells for vdw or metal forces
no vdw	Ignore all short range (non-bonded) interactions
optim energy *f*	Relax structure based on energy, force or position, with:
optim force *f*	*f* = permitted variation (tolerance) (DL_POLY units).
optim position *f*	
pres *f*	Set required system pressure to *f* k-atm (target pressure for constant pressure ensembles)
prim *f*	Set primary cutoff to *f* (Å) (for multiple timestep algorithm only)
print *n*	Print system data every *n* timesteps
print rdf	Print radial distribution functions
put shells on cores	Superimpose electrostatic shells and cores at start
quaternion *f*	Set quaternion tolerance to *f* (default 10^{-8})
rdf *n w*	Calculate radial distribution functions with: *n* the time step interval between configurations *w* the RDF bin width ($\mathring{A}$). (Note: $range = r_{cut}$).
reaction	Select reaction field electrostatics
reaction precision *f*	Select damped reaction field electrostatics with automatic parameter optimisation ($0 < f < 1.E-4$)
reaction damped *f*	Select damped reaction field electrostatics with user chosen damping parameter *f* ($Å^{-1}$)
regauss *n*	Reset velocities at timestep interval *n* using Gaussian distribution.
restart	Restart job from end point of previous run (i.e. continue current simulation)
restart noscale	Restart job from previous run with no temperature scaling (i.e. begin a new simulation from older run)
restart scale	Restart job from previous run with temperature scaling (i.e. begin a new simulation from older run)
rlxtol *f*	Reset force tolerance for shell relaxation to *f* = (DL_POLY units), (1.0 default).
rvdw *f*	Set required vdw forces cutoff to *f* (Å)
scale *n*	Rescale atomic velocities every *n* steps (during equilibration) using *ad hoc* rescaling.
shake *f*	Set shake tolerance to *f* (default 10^{-8})
shift	Calculate electrostatic forces using shifted coulombic potential
shift precision *f*	Select damped shifted coulombic force electrostatics with automatic parameter optimisation ($0 < f < 1.E-4$)
shift damped *f*	Select damped shifted coulombic force electrostatics with user chosen damping parameter *f* ($Å^{-1}$)
spme precision *f*	Select SPME for electrostatics, with automatic parameter optimisation ($0 < f < .5$)
spme sum *α k1 k2 k3*	Select SPME for electrostatics, with: α = Ewald convergence parameter ($Å^{-1}$)

	k1 = maximum k-vector index in x-direction
	k2 = maximum k-vector index in y-direction
	k3 = maximum k-vector index in z-direction
stack *n*	Set rolling average stack to *n* timesteps
stats *n*	Accumulate statistics data every *n* timesteps
steps *n*	Run simulation for *n* timesteps
temp *f*	Set required simulation temperature to *f* K
traj *i j k*	Write HISTORY file with controls:
	i = start timestep for dumping configurations
	j = timestep interval between configurations
	k = data level (i.e. variable `keytrj` see table 4.3)
timestep *f*	Set timestep to *f* ps
zden *n w z*	Calculate the z-density profile with:
	n the time step interval between configurations
	w the z-density bin width ($\AA$)
	z the range of z coordinate required (-z/2,+z/2) ($\AA$)
zero	Perform zero temperature MD run

4.1.1.3 Further Comments on the CONTROL File

1. A number of the directives (or their **mutually exclusive** alternatives) are **mandatory**:
   (a) **timestep**: specifying the simulation timestep;
   (b) **temp** or **zero** : specifying the system temperature (not mutually exclusive);
   (c) **ewald sum** or **ewald precision** or **spme sum** or **spme precision** or **hke sum** or **hke precision** or **coul** or **shift** or **distan** or **reaction** or **no elec**: specifying the required coulombic forces option;
   (d) **cut** and **delr**: specifying the short range forces cutoff and Verlet strip;
   (e) **prim**: specifying primary forces cutoff (**if mult>2 only**).

2. The **job time** and **close time** directives are required to ensure a controlled close down procedure when a job runs out of time. The time specified by the **job time** directive indicates the total time allowed for the job. (This must obviously be set equal to the time specified to the operating system when the job is submitted.) The **close time** directive represents the time DL_POLY Classic will require to write and close all the data files at the end of processing. This means the *effective* processing time limit is equal to the job time minus the close time. Thus when DL_POLY Classic reaches the effective job time limit it begins the close down procedure with enough time in hand to ensure the files are correctly written. In this way you may be sure the restart files etc. are complete when the job terminates. Note that setting the close time too small will mean the job will crash before the files have been finished. If it is set too large DL_POLY Classic will begin closing down too early. How large the close time needs to be to ensure safe close down is system dependent and a matter of experience. It generally increases with the job size.

3. Note that the default time unit for **job time** is seconds, however this may be changed by addition of an extra character after the number on the the directive line. Thus *m* will set it to minutes; *h* to hours; and *d* to days. You can even skip the number altogether and put *indef*, which will set the default job time to 1 million years, which should be enough for anyone.

Note however that you will lose the capability to end the job within the specified close time, so you should be sure the job will finish without crashing.

4. The starting options for a simulation are governed by the keyword **restart**. If this is **not** specified in the CONTROL file, the simulation will start as new. If specifed, it will either **continue** a previous simulation (**restart**) or start a **new simulation** with initial temperature scaling of the previous configuration (**restart scale**) or without initial temperature scaling (**restart noscale**). Internally these options are handled by the integer variable `keyres`, which is explained in table 4.1.

5. The various **ensemble** options (i.e. **nve**, **nvt ber**, **nvt evans, nvt hoover, npt ber, npt hoover**, **nst ber**, **nst hoover**) are mutually exclusive, though none is mandatory (the default is the NVE ensemble). These options are handled internally by the integer variable `keyens`. The meaning of this variable is explained in table 4.2.

6. The force selection directives **ewald sum**, **ewald precision**, **reaction**, **coul**, **shift**, **dist**, **no elec** and **no vdw** are handled internally by the integer variable `keyfce`. See table 4.4 for an explanation of this variable. Note that these options are mutually exclusive.

7. The choice of reaction field electrostatics (directive **reaction**) requires also the specification of the relative dielectric constant external to the cavity. This is specified in the **eps** directive.

8. DL_POLY Classic uses as many as three different potential cutoffs. These are as follows:

   (a) `cut` - this is the universal cutoff. It applies to the real space part of the electrostatics calculations and to the van der Waals potentials if no other cutoff is applied;

   (b) `rvdw` - this is the user-specified cutoff for the van der Waals potentials. If not specified its value defaults to `rcut`. It cannot exceed `cut`;

   (c) `rprim` - this is used in the multiple timestep algorithm to specify the primary atom region (see section 2.5.8). It is ignored if the multiple timestep option is not used.

9. Some directives are optional. If not specified DL_POLY Classic will take default values if necessary. The defaults appear in the above table.

10. The **zero** directive enables a zero temperature simulation. This is intended as a crude energy minimiser to help relax a system before a simulation begins. It should not be thought of as a true energy minimisation method.

11. The **optim** directive enables a conjugate gradient energy minimisation of the configuration. There are three additional options: **energy**, **force** and **position**, which decide convergence on the basis of energy, force or position. The recommended option is **force** which is suitable for most cases. Note that the additional parameter the user must supply is the tolerance for the convergence. This must be appropriate for the chosen option. All are expressed in DL_POLY internal units. For the **force** option a value of about 1.0 is appropriate for many cases.

12. The **minim** directive enables a programmed minimisation, which combines conjugate gradient minimisation with a molecular dynamics search. The three additional options: **energy**, **force** and **position** refer to the CG minimisation, as described for the **optim** directive above. The user must also supply an integer number of time steps for the interval between successive CG minimisations and the convergence tolerance for each minimisation. The tolerance is expressed in the appropriate internal units.

13. The DL_POLY Classic multiple timestep option is invoked if the number appearing with the **mult** directive is greater than 2. This number (stored in the variable `multt`) specifies the number of timesteps (the multi-step) that elapse between partitions of the full Verlet neighbour list into primary and secondary atoms.

14. If a multiple time-step is used, (i.e. `multt`>2), then statistics for radial distribution functions are collected only at updates of the secondary neighbour list. The number specified on the **rdf** directive (the variable `nstbgr`) means that RDF data are accumulated at intervals of `nstbgr`×`multt` timesteps.

15. As a default, DL_POLY Classic does not store statistical data during the equilibration period. If the directive **collect** is used, equilibration data will be incorporated into the overall statistics.

16. The directive **delr** specifies the width of the border to be used in the Verlet neighbour list construction. The width is stored in the variable `delr`. The list is updated whenever two or more atoms have moved a distance of more then `delr`/2 from their positions at the last update of the Verlet list.

17. The directive **impact** is intended to simulate the effects of a high energy atomsic impact, such as occurs in radiation damage. The user must supply the (integer) identity of the impacted atom, the time step when the impact takes place (usually after equilibration), the recoil energy of the impacted atom (in *kilo electron volts*), and the direction of the recoil (i.e. three components of a unit vector specifying the direction).

18. The directive **no fic** activates and option that tries to prevent the occurrence of the flying ice cube that sometimes arises in long time simulations using the Berendsen method for thermostating. This arises through accumulated numerical round-off, which gradually transfers momentum from the system kinetic energy to the centre-of-mass momentum, resulting in a zero Kelvin structure with a net linear momentum. This option removes the COM momentum at user selected intervals.

19. The **put shells on cores** directive ensures that associated cores and shells start the simulation at exactly the same place. It is not usual to do this however.

Table 4.1: Internal Restart Key

keyres	meaning
0	start new simulation from CONFIG file, and assign velocities from Gaussian distribution.
1	continue current simulation
2	start new simulation from CONFIG file, and rescale velocities to desired temperature
3	start new simulation from CONFIG file, without rescaling the velocities

Table 4.2: Internal Ensemble Key

keyens	meaning
0	Microcanonical ensemble (NVE)
1	Evans NVT ensemble
2	Berendsen NVT ensemble
3	Nosé-Hoover NVT ensemble
4	Berendsen NPT ensemble
5	Nosé-Hoover NPT ensemble
6	Berendsen N$\underline{\underline{\sigma}}$T ensemble
7	Nosé-Hoover N$\underline{\underline{\sigma}}$T ensemble
8	Potential of mean force (NVE) ensemble

Table 4.3: Internal Trajectory File Key

keytrj	meaning
0	coordinates only in file
1	coordinates and velocities in file
2	coordinates, velocities and forces in file

Table 4.4: Non-bonded force key

keyfce	meaning
odd	evaluate short-range potentials and electrostatics
even	evaluate Electrostatic potential only
Electrostatics are evaluated as follows:	
0†, 1‡	Ignore Electrostatic interactions
2, 3	Ewald summation
4, 5	distance dependent dielectric
6, 7	standard truncated Coulombic potential
8, 9	truncated and shifted Coulombic potential
10,11	Reaction Field electrostatics
12,13	SPME electrostatics
14,15	Hautman-Klein Ewald electrostatics

† `keyfce` = 0 means no non-bonded terms are evaluated.
‡ `keyfce` = 1 means only short-range potentials are evaluated.

4.1.2 The CONFIG File

The CONFIG file contains the dimensions of the unit cell, the key for periodic boundary conditions and the atomic labels, coordinates, velocities and forces. This file is read by the subroutine SYSGEN. (It is also read by the subroutine SIMDEF if the **ewald precision** directive is used.) The first few records of a typical CONFIG file are shown below:

```
Lennard-Jones Argon
         2         3       255  -176595.066855
     21.023998260000      0.000000000000      0.000000000000
      0.000000000000     21.023998260000      0.000000000000
      0.000000000000      0.000000000000     21.023998260000
Ar              1
     7.798997031         2.409934763         5.506441637
  -2.12339759919      -1.85576903413      0.125163024806
  -417.940093856       292.432569373      -472.434039806
Ar              2
     2.821617729        0.7180021261         7.417288159
   1.07786776343      0.168433841280E-01 -0.269392807911
  -188.920889755      -413.545510271       294.149380530
Ar              3
    -8.113009749         2.773816641         5.199345225
  0.388066563418      -1.37628108908      -1.24723236452
   608.168259627      -422.414753563      -250.737138386
Ar              4
    -10.31216635         2.857971798         8.090920140
   1.76536230573      -1.58904200978      -2.48066272817
  -66.0234000384      -47.6492437764       90.0074615387

etc.
```

4.1.2.1 Format

The file is fixed-formatted: integers as "i10", reals as "f20.0". The header record is formatted as 80 alphanumeric characters.

4.1.2.2 Definitions of Variables

record 1		
`header`	a80	title line
record 2		
`levcfg`	integer	CONFIG file key. See table 4.5 for permitted values
`imcon`	integer	Periodic boundary key. See table 4.6 for permitted values
`natms`	integer	Number of atoms in file
`engcfg`	real	Configuration energy in DL_POLY units
record 3	omitted if `imcon` = 0	
`cell(1)`	real	x component of a cell vector
`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
record 4	omitted if `imcon` = 0	
`cell(4)`	real	x component of b cell vector

`cell(5)`	real	y component of b cell vector
`cell(6)`	real	z component of b cell vector
record 5	omitted if `imcon` = 0	
`cell(7)`	real	x component of c cell vector
`cell(8)`	real	y component of c cell vector
`cell(9)`	real	z component of c cell vector

Subsequent records consists of blocks of between 2 and 4 records depending on the value of the `levcfg` variable. Each block refers to one atom. The atoms must be listed sequentially in order of increasing index. Within each block the data are as follows:

record i		
`atmnam`	a8	atom name.
`index`	integer	atom index
`atmnum`	integer	atomic number
record ii		
`xxx`	real	x coordinate
`yyy`	real	y coordinate
`zzz`	real	z coordinate
record iii	included only if `levcfg` > 0	
`vxx`	real	x component of velocity
`vyy`	real	y component of velocity
`vzz`	real	x component of velocity
record iv	included only if `levcfg` > 1	
`fxx`	real	x component of force
`fyy`	real	y component of force
`fzz`	real	z component of force

Note that on **record i** only the atom name is mandatory, any other items are not read by DL_POLY Classic but may be added to aid alternative uses of the file, for example the DL_POLY Classic Graphical User Interface [9].

4.1.2.3 Further Comments

The CONFIG file has the same format as the output files REVCON (section 4.2.3) and CFGMIN (section 3.2.4). When restarting from a previous run of DL_POLY Classic (i.e. using the **restart**, **restart scale** or **restart noscale** directives in the CONTROL file - above), the CONFIG file must be replaced by the REVCON file, which is renamed as the CONFIG file. The *copy* macro in the *execute* sub-directory of DL_POLY Classic does this for you.

Table 4.5: CONFIG file key (record 2)

levcfg	meaning
0	Coordinates included in file
1	Coordinates and velocities included in file
2	Coordinates, velocities and forces included in file

Table 4.6: Periodic boundary key (record 2)

imcon	meaning
0	no periodic boundaries
1	cubic boundary conditions
2	orthorhombic boundary conditions
3	parallelepiped boundary conditions
4	truncated octahedral boundary conditions
5	rhombic dodecahedral boundary conditions
6	x-y parallelogram boundary conditions with no periodicity in the z direction
7	hexagonal prism boundary conditions

4.1.3 The FIELD File

The FIELD file contains the force field information defining the nature of the molecular forces. It is read by the subroutine SYSDEF. Excerpts from a force field file are shown below. The example is the antibiotic Valinomycin in a cluster of 146 water molecules.

```
Valinomycin Molecule with 146 SPC Waters
UNITS kcal

MOLECULES    2
Valinomycin
NUMMOLS 1
ATOMS 168
    O         16.0000     -0.4160    1
    OS        16.0000     -0.4550    1
    "           "           "        "
    "           "           "        "
    HC         1.0080      0.0580    1
    C         12.0100      0.4770    1
BONDS 78
harm   31   19 674.000     1.44900
harm   33   31 620.000     1.52600
    "    "    "    "         "
    "    "    "    "         "
harm  168   19 980.000     1.33500
harm  168  162 634.000     1.52200
CONSTRAINTS 90
   20   19    1.000017
   22   21    1.000032
    "    "     "
    "    "     "
  166  164    1.000087
  167  164    0.999968
ANGLES 312
harm   43    2   44  200.00      116.40
harm   69    5   70  200.00      116.40
  "    "    "    "     "           "
  "    "    "    "     "           "
harm   18  168  162  160.00      120.40
harm   19  168  162  140.00      116.60
DIHEDRALS 371
harm    1   43    2   44  2.3000      180.00
harm   31   43    2   44  2.3000      180.00
  "    "    "    "    "     "           "
  "    "    "    "    "     "           "
cos   149   17  161   16  10.500      180.00
cos   162   19  168   18  10.500      180.00
FINISH
SPC Water
NUMMOLS 146
```

```
ATOMS  3
    OW       16.0000     -0.8200
    HW        1.0080      0.4100
    HW        1.0080      0.4100
CONSTRAINTS  3
    1    2    1.0000
    1    3    1.0000
    2    3    1.63299
FINISH
VDW   45
C       C         lj   0.12000      3.2963
C       CT        lj   0.08485      3.2518
"       "         "     "            "
"       "         "     "            "
"       "         "     "            "
OW      OS        lj   0.15100      3.0451
OS      OS        lj   0.15000      2.9400
CLOSE
```

4.1.3.1 Format

The FIELD file is free formatted (though it should be noted that atom names are limited to 8 characters and potential function keys are a maximum of 4 characters). The contents of the file are variable and are defined by the use of **directives**. Additional information is associated with the directives. The file is not case sensitive.

4.1.3.2 Definitions of Variables

The file divides into three sections: general information, molecular descriptions, and non-bonded interaction descriptions, appearing in that order in the file.

4.1.3.2.1 General information

The first record in the FIELD file is the title. It must be followed by the **units** directive. Both of these are mandatory. These records may optionally be followed by the **neut** directive.

record 1		
header	a80	field file header
record 2		
units	a40	Unit of energy used for input and output
record 3 (optional)		
neut	a40	activate the neutral/charge groups option for the electrostatic calculations

The energy units on the **units** directive are described by additional keywords:

a eV, for electron-volts

b kcal, for k-calories mol^{-1}

c kJ, for k-Joules mol^{-1}

d K, for Kelvin

e internal, for DL_POLY Classic internal units (10 J mol^{-1}).

If no units keyword is entered, DL_POLY Classic units are assumed for both input and output. The units keyword may appear anywhere on the data record provided it does not exceed column 40. The **units** directive only affects the input and output interfaces, all internal calculations are handled using DL_POLY Classic units.

4.1.3.2.2 Molecular details

It is important for the user to understand that there is an structural correspondence between the FIELD file and the CONFIG file described above. It is required that the order of specification of molecular types and their atomic constituents in the FIELD file follows the order in which they appear in the CONFIG file. Failure to adhere to this common sequence will be detected by DL_POLY Classic and result in premature termination of the job. It is therefore essential to work from the CONFIG file when constructing the FIELD file. It is not as difficult as it sounds!

The entry of the molecular details begins with the mandatory directive:

molecules *n*

where *n* is an integer specifying the number of different *types* of molecule appearing in the FIELD file. Once this directive has been encountered, DL_POLY Classic enters the *molecular description* environment in which only molecular decription keywords and data are valid.

Immediately following the **molecules** directive, are the records defining individual molecules:

1. *name-of-molecule*
   which can be any character string up to 80 characters in length. (Note: this is not a directive, just a simple character string.)

2. **nummols *n***
   where *n* is the number of times a molecule of this type appears in the simulated system. The molecular data then follow in subsequent records:

3. **atoms *n***
   where *n* indicates the number of atoms in this type of molecule. A number of records follow, each giving details of the atoms in the molecule i.e. site names, masses and charges. Each record carries the entries:

`sitnam`	a8	atomic site name
`weight`	real	atomic site mass
`chge`	real	atomic site charge
`nrept`	integer	repeat counter
`ifrz`	integer	'frozen' atom (if `ifrz`> 0)
`igrp`	integer	neutral/charge group number

   Note that these entries are order sensitive. Do not leave blank entries unless all parameters appearing after the last specified are void. The integer `nrept` need not be specified (in which case a value of 1 is assumed.) A number greater than 1 specified here indicates that the next

(`nrept` - 1) entries in the CONFIG file are ascribed the atomic characteristics given in the current record. The sum of the repeat numbers for all atoms in a molecule should equal the number specified by the **atoms** directive.

4. **shell *n m***
where n is the number of core-shell units and m is an integer specifying which shell model is required:

   - m=1 for adiabatic shell model;
   - m=2 for relaxed shell model;

   Each of the subsequent n records contains:

index 1	integer	site index of core
index 2	integer	site index of shell
k	real	force constant of core-shell spring
k_4	real	quartic (anharmonic) force constant of spring

   The spring force constant k is entered in units of `engunit` Å^{-2}, (or `engunit` Å^{-4} for k_4), where `engunit` is the energy unit specified in the **units** directive. The general spring potential has the form

$$V_{spring}(r_{ij}) = \frac{1}{2}kr_{ij}^2 + \frac{1}{4}k_4 r_{ij}^4,$$

   where usually $k >> k_4$.

   The adiabatic and relaxed shell models are mutually exclusive options in the same simulation.

   Note that the atomic site indices referred to in this table are indices arising from numbering each atom in the molecule from 1 to the number specified in the **atoms** directive for this molecule. This same numbering scheme should be used for all descriptions of this molecule, including the **bonds**, **constraints**, **angles**, and **dihedrals** entries described below. DL_POLY Classic will itself construct the global indices for all atoms in the systems.

   This directive (and associated data records) need not be specified if the molecule contains no core-shell units.

5. **bonds *n***
where n is the number of flexible chemical bonds in the molecule. Each of the subsequent n records contains:

bond key	a4	see table 4.7
index 1	integer	first atomic site in bond
index 2	integer	second atomic site in bond
variable 1	real	potential parameter see table 4.7
variable 2	real	potential parameter see table 4.7
variable 3	real	potential parameter see table 4.7
variable 4	real	potential parameter see table 4.7

   The meaning of these variables is given in table 4.7. This directive (and associated data records) need not be specified if the molecule contains no flexible chemical bonds. See the note on the atomic indices appearing under the **shell** directive above.

Table 4.7: Chemical bond potentials

key	potential type	Variables (1-4)				functional form
harm **-hrm**	Harmonic	k	r_0			$U(r) = \frac{1}{2}k(r-r_0)^2$
mors **-mrs**	Morse	E_0	r_0	k		$U(r) = E_0[\{1-\exp(-k(r-r_0))\}^2 - 1]$
12-6 **-126**	12-6	A	B			$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^6}\right)$
rhrm **-rhm**	Restraint	k	r_0	r_c		$U(r) = \frac{1}{2}k(r-r_0)^2 \quad \lvert r-r_0\rvert \le r_c$ $U(r) = \frac{1}{2}kr_c^2 + kr_c(\lvert r-r_0\rvert - r_c) \quad \lvert r-r_0\rvert > r_c$
quar **-qur**	Quartic	k	r_0	k'	k''	$U(r) = \frac{k}{2}(r-r_0)^2 + \frac{k'}{3}(r-r_0)^3 + \frac{k''}{4}(r-r_0)^4$
buck **-bck**	Buckingham	A	ρ	C		$U(r) = Aexp(-r/\rho) - C/r^6$
fene **-bck**	FENE	k	R_o	Δ		$U(r_{ij}) = -0.5\ k\ R_o^2\ ln\left[1 - \left(\frac{r_{ij}-\Delta}{R_o}\right)^2\right]$
coul **-cou**	Coulombic	q_i	q_j			$U(r_{ij}) = \frac{1}{4\pi\epsilon}\frac{q_iq_j}{r_{ij}}$

Note: bond potentials with a dash (-) as the first character of the keyword, do not contribute to the *excluded atoms list* (see section 2.1). In this case DL_POLY Classic will also calculate the nonbonded pair potentials between the described atoms, unless these are deactivated by another potential specification.

6. **constraints** ***n***
   where n is the number of constraint bonds in the molecule. Each of the following n records contains:

index 1	integer	first atomic index
index 2	integer	second atomic index
bondlength	real	constraint bond length

   This directive (and associated data records) need not be specified if the molecule contains no constraint bonds. See the note on the atomic indices appearing under the **shell** directive above.

7. **pmf** ***b***

where b is the potential of mean force bondlength (Å). There follows the definitions of two PMF units:

(a) **pmf unit** n_1
where n_1 is the number of sites in the first unit. The subsequent n_1 records provide the site indices and weighting. Each record contains:

index	integer	atomic site index
weight	real	site weighting

(b) **pmf unit** n_2
where n_2 is the number of sites in the second unit. The subsequent n_2 records provide the site indices and weighting. Each record contains:

index	integer	atomic site index
weight	real	site weighting

This directive (and associated data records) need not be specified if no PMF constraints are present. See the note on the atomic indices appearing under the **shell** directive above. The pmf bondlength applies to the distance between the centres of the two pmf units. The centre, $\underline{R}$, of each unit is given by

$$\underline{R} = \frac{\sum_\alpha w_\alpha \underline{r}_\alpha}{\sum_\alpha w_\alpha}$$

where $\underline{r}_\alpha$ is a site position and w_α the site weighting. Note that the pmf constraint is intramolecular. To define a constraint between two molecules, the molecules must be described as part of the same DL_POLY "molecule". This is illustrated in test case 6, where a pmf constraint is imposed between a potassium ion and the centre of mass of a water molecule. DL_POLY Classic allows only one type of pmf constraint per system. The value of `nummols` for this molecule determines the number of pmf constraint in the system.

Note that the directive **ensemble pmf** must be specified in the CONTROL file for this option to be implemented correctly.

8. **angles** ***n***
where n is the number of valence angle bonds in the molecule. Each of the n records following contains:

angle key	a4	potential key. See table 4.8
index 1	integer	first atomic index
index 2	integer	second atomic index (central site)
index 3	integer	third atomic index
variable 1	real	potential parameter see table 4.8
variable 2	real	potential parameter see table 4.8
variable 2	real	potential parameter see table 4.8
variable 3	real	potential parameter see table 4.8
variable 4	real	potential parameter see table 4.8

The meaning of these variables is given in table 4.8. See the note on the atomic indices appearing under the **shell** directive above. This directive (and associated data records) need not be specified if the molecule contains no angular terms.

Table 4.8: Valence Angle potentials

key	potential type	Parameters p_1-p_4				functional form†
harm **-hrm**	Harmonic	k	θ_0			$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2$
quar **-qur**	Quartic	k	θ_0	k'	k''	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 + \frac{k'}{3}(\theta - \theta_0)^3 + \frac{k''}{4}(\theta - \theta_0)^4$
thrm **-thm**	Truncated harmonic	k	θ_0	ρ		$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
shrm **-shm**	Screened harmonic	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs1 **-bv1**	Screened Vessal[28]	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{8(\theta-\theta_0)^2}\left\{[(\theta_0 - \pi)^2 - (\theta - \pi)^2]^2\right\}$ $\exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs2 **-bv2**	Truncated Vessal[29]	k	θ_0	a	ρ	$U(\theta) = k[\theta^a(\theta - \theta_0)^2(\theta + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1}$ $(\theta - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
hcos **-hcs**	Harmonic Cosine	k	θ_0			$U(\theta) = \frac{k}{2}(cos(\theta) - cos(\theta_0))^2$
cos **-cos**	Cosine	A	δ	m		$U(\theta) = A[1 + cos(m\theta - \delta)]$
mmsb **-msb**	MM Stretch-bend	A	θ_0	d_{ab}	d_{ac}	$U(\theta) = A(\theta - \theta_0)(r_{ab} - d_{ab})(r_{ac} - d_{ac})$
stst **-sts**	Compass stretch-stretch	A	d_{ab}	d_{ac}		$U_{bac} = A(r_{ab} - d_{ab})(r_{ac} - d_{ac})$
stbe **-stb**	Compass stretch-bend	A	θ_0	d_{ab}		$U_{bac} = A(\theta - \theta_0)(r_{ab} - d_{ab})$
cmps **-cmp**	Compass all terms	A	B	C	θ_0	$U_{bac} = A(r_{ab} - d_{ab})(r_{ac} - d_{ac}) + (\theta - \theta_0)*$ $(B(r_{ab} - d_{ab}) + C(r_{ac} - d_{ac}))$
		$p_5 = d_{ab}$		$p_6 = d_{ac}$		

†θ is the a-b-c angle.

Note: valence angle potentials with a dash (-) as the first character of the keyword, do not contribute to the *excluded atoms list* (see section 2.1). In this case DL_POLY Classic will calculate the nonbonded pair potentials between the described atoms.

9. **dihedrals *n***
where n is the number of dihedral interactions present in the molecule. Each of the following n records contains:

dihedral key	a4	potential key. See table 4.9
index 1	integer	first atomic index
index 2	integer	second atomic index
index 3	integer	third atomic index
index 4	integer	fourth atomic index
variable 1	real	potential parameter see table 4.9
variable 2	real	potential parameter see table 4.9
variable 3	real	potential parameter see table 4.9
variable 4	real	1-4 electrostatic interaction scale factor.
variable 5	real	1-4 Van der Waals interaction scale factor.

The meaning of the variables 1-3 is given in table 4.9. The variables 4 and 5 specify the scaling factor for the 1-4 electrostatic and Van der Waals nonbonded interactions respectively. This directive (and associated data records) need not be specified if the molecule contains no dihedral angle terms. See the note on the atomic indices appearing under the **shell** directive above.

Table 4.9: Dihedral Angle Potentials

key	potential type	Variables (1-4)				functional form‡
cos	Cosine	A	δ	m		$U(\phi) = A\,[1 + \cos(m\phi - \delta)]$
harm	Harmonic	k	ϕ_0			$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0			$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
cos3	Triple cosine	A_1	A_2	A_3		$U(\phi) = \frac{1}{2}A_1(1 + cos(\phi)) + \frac{1}{2}A_2(1 - cos(2\phi))$ $+\frac{1}{2}A_3(1 + cos(3\phi))$
ryck	Ryckaert-Bellemans	A				$U(\phi) = A(a_0 + a_1 cos\phi - a_2 cos^2\phi + a_3 cos^3\phi +$ $a_4 cos^4\phi + a_5 cos^5\phi)$ $[a_0 \rightarrow a_5$ pre-set]
rbf	Fluorinated Ryckaert-Bellemans	B				$U(\phi) = B(b_0 - b_1 cos\phi - b_2 cos^2\phi - b_3 cos^3\phi +$ $b_4 cos^4\phi + b_5 cos^5\phi + b_6 exp(-b_7(\phi - \pi))$ $[b_0 \rightarrow b_6$ pre-set]
opls	OPLS	A_0	A_1	A_2	A_3	$U(\phi) = A_0 + \frac{1}{2}(A_1(1 + cos(\phi)) + A_2(1 - cos(2\phi))$ $+A_3(1 + cos(3\phi)))$

‡ϕ is the a-b-c-d dihedral angle.

10. **inversions *n***
where n is the number of inversion interactions present in the molecule. Each of the following

n records contains:

inversion key	a4	potential key. See table 4.10
index 1	integer	first atomic index
index 2	integer	second atomic index
index 3	integer	third atomic index
index 4	integer	fourth atomic index
variable 1	real	potential parameter see table 4.10
variable 2	real	potential parameter see table 4.10

The meaning of the variables 1-2 is given in table 4.10. This directive (and associated data records) need not be specified if the molecule contains no inversion angle terms. See the note on the atomic indices appearing under the **shell** directive above.

Table 4.10: Inversion Angle Potentials

key	potential type	Variables (1-2)		functional form‡
harm	Harmonic	k	ϕ_0	$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0	$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
plan	Planar	A		$U(\phi) = A\,[1 - \cos(\phi)]$
calc	Calcite	A	B	$U(u) = Au^2 + Bu^4$

‡ϕ is the inversion angle.

Note that the calcite potential is not dependent on an angle ϕ, but on a *displacement* u. See section 2.2.8 for details.

11. **rigid *n***
where n is the number of rigid units in the molecule. It is followed by at least n records, each specifying the sites in a rigid unit:

m	integer	number of sites in rigid unit
site 1	integer	first site atomic index
site 2	integer	second site atomic index
site 3	integer	third site atomic index
..	..	*etc.*
site m	integer	m'th site atomic index

Up to 15 sites can be specified on the first record. Additional records are used if necessary. Up to 16 sites are specified per record thereafter.

This directive (and associated data records) need not be specified if the molecule contains no rigid units. See the note on the atomic indices appearing under the **shell** directive above.

12. **teth *n***
where n is the number of tethered atoms in the molecule. It is followed by n records specifying the tethered sites in the molecule:

tether key	a4	tethering potential key see table 4.11
index	integer	atomic index
variable 1	real	potential parameter see table 4.11
variable 2	real	potential parameter see table 4.11
variable 3	real	potential parameter see table 4.11
variable 4	real	potential parameter see table 4.11

This directive (and associated data records) need not be specified if the molecule contains no tethered atoms. See the note on the atomic indices appearing under the **shell** directive above.

Table 4.11: Tethering potentials

key	potential type	Variables (1-3)			functional form
harm	Harmonic	k			$U(r) = \frac{1}{2}kr^2$
rhrm	Restraint	k	r_c		$U(r) = \frac{1}{2}kr^2 \qquad r \le r_c$ $U = \frac{1}{2}kr_c^2 + kr_c(r - r_c) \qquad r > r_c$
quar	Quartic	k	k'	k''	$U(r) = \frac{k}{2}r^2 + \frac{k'}{3}r^3 + \frac{k''}{4}r^4$

13. **finish**
This directive is entered to signal to DL_POLY Classic that the entry of the details of a molecule has been completed.

The entries for a second molecule may now be entered, beginning with the *name-of-molecule* record and ending with the **finish** directive.

The cycle is repeated until all the types of molecules indicated by the **molecules** directive have been entered.

The user is recommended to look at the example FIELD files in the *data* directory to see how typical FIELD files are constructed.

4.1.3.3 Non-bonded Interactions

Non-bonded interactions are identified by atom types as opposed to specific atomic indices. The first type of non-bonded potentials are the pair potentials. The input of pair potential data is signalled by the directive:

vdw *n*

where n is the number of pair potentials to be entered. There follows n records, each specifying a particular pair potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
key	a4	potential key. See table 4.12
variable 1	real	potential parameter see table 4.12
variable 2	real	potential parameter see table 4.12
variable 3	real	potential parameter see table 4.12
variable 4	real	potential parameter see table 4.12
variable 5	real	potential parameter see table 4.12

The variables pertaining to each potential are described in table 4.12. Note that any pair potential not specified in the FIELD file, will be assumed to be zero.

The specification of three body potentials is initiated by the directive:

tbp *n*

where n is the number of three-body potentials to be entered. There follows n records, each specifying a particular three body potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type (central site)
atmnam 3	a8	third atom type
key	a4	potential key. See table 4.13
variable 1	real	potential parameter see table 4.13
variable 2	real	potential parameter see table 4.13
variable 3	real	potential parameter see table 4.13
variable 4	real	potential parameter see table 4.13
variable 5	real	cutoff range for this potential (Å)

The variables pertaining to each potential are described in table 4.13. Note that the fifth variable is the range at which the three body potential is truncated. The distance is in Å, measured from the central atom.

The specification of four body potentials is initiated by the directive:

fbp *n*

where n is the number of four-body potentials to be entered. There follows n records, each specifying a particular four-body potential in the following manner:

atmnam 1	a8	first atom type (central site)
atmnam 2	a8	second atom type
atmnam 3	a8	third atom type
atmnam 4	a8	fourth atom type

Table 4.12: Definition of pair potential functions and variables

key	potential type	Variables (1-5)					functional form
12-6	12-6	A	B				$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^6}\right)$
lj	Lennard-Jones	ϵ	σ				$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right]$
nm	n-m	E_o	n	m	r_0		$U(r) = \frac{E_o}{(n-m)}\left[m\left(\frac{r_o}{r}\right)^n - n\left(\frac{r_o}{r}\right)^m\right]$
buck	Buckingham	A	ρ	C			$U(r) = A\ \exp\left(-\frac{r}{\rho}\right) - \frac{C}{r^6}$
bhm	Born-Huggins -Meyer	A	B	σ	C	D	$U(r) = A\ \exp[B(\sigma - r)] - \frac{C}{r^6} - \frac{D}{r^8}$
hbnd	12-10 H-bond	A	B				$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^{10}}\right)$
snm	Shifted force† n-m [32]	E_o	n	m	r_0	r_c‡	$U(r) = \frac{\alpha E_o}{(n-m)}\times$ $\left[m\beta^n\left\{\left(\frac{r_o}{r}\right)^n - \left(\frac{1}{\gamma}\right)^n\right\} - n\beta^m\left\{\left(\frac{r_o}{r}\right)^m - \left(\frac{1}{\gamma}\right)^m\right\}\right]$ $+\frac{nm\alpha E_o}{(n-m)}\left(\frac{r-\gamma r_o}{\gamma r_o}\right)\left\{\left(\frac{\beta}{\gamma}\right)^n - \left(\frac{\beta}{\gamma}\right)^m\right\}$
mors	Morse	E_0	r_0	k			$U(r) = E_0[\{1 - \exp(-k(r - r_0))\}^2 - 1]$
wca	WCA	ϵ	σ				$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] + \epsilon \qquad (r < \sigma * 2^{1/6})$
tab	Tabulation						tabulated potential (see section 4.1.5

† Note: in this formula the terms α, β and γ are compound expressions involving the variables E_o, n, m, r_0 and r_c. See section 2.3.1 for further details.
‡ Note: r_c defaults to the general van der Waals cutoff (`rvdw` or `rcut`) if it is set to zero or not specified in the FIELD file.

key	a4	potential key. See table 4.14
variable 1	real	potential parameter see table 4.14
variable 2	real	potential parameter see table 4.14
variable 3	real	cutoff range for this potential (Å)

The variables pertaining to each potential are described in table 4.14. Note that the third variable is the range at which the four-body potential is truncated. The distance is in Å, measured from the central atom.

4.1.3.4 Metal Potentials

Metal potentials in DL_POLY Classic are based on the embedded atom model (EAM) [35, 36] and the Finnis-Sinclair model (FSM)[3] .

Table 4.13: Three-body potentials

key	potential type	Variables (1-4)				functional form†
thrm	Truncated harmonic	k	θ_0	ρ		$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
shrm	Screened harmonic	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs1	Screened Vessal[28]	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{8(\theta-\theta_0)^2}\left\{[(\theta_0 - \pi)^2 - (\theta - \pi)^2]^2\right\}$ $\exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs2	Truncated Vessal[29]	k	θ_0	a	ρ	$U(\theta) = k[\theta^a(\theta - \theta_0)^2(\theta + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1}$ $(\theta - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
hbnd	H-bond [7]	D_{hb}	R_{hb}			$U(\theta) = D_{hb}cos^4(\theta)[5(R_{hb}/r_{jk})^{12} - 6(R_{hb}/r_{jk})^{10}]$

†θ is the a-b-c angle.

Table 4.14: Four-body Potentials

key	potential type	Variables (1-2)		functional form‡
harm	Harmonic	k	ϕ_0	$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0	$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
plan	Planar	A		$U(\phi) = A\,[1 - \cos(\phi)]$

‡ϕ is the inversion angle.

The EAM potentials are tabulated and are supplied to DL_POLY Classic in the input file TABEAM (see 4.1.6).

The FSM potentials are analytical and DL_POLY Classic supports the explicit forms due to: Finnis and Sinclair [3] ; Sutton and Chen [38, 39] ; and Gupta [41] ;

Metal potentials, like van der Waals potentials are also non-bonded potentials and are characterised by atom types rather than specific atomic indices. The input of metal potential data is signalled by the directive:

metal *n*

where n is the number of metal potentials to be entered. There follows n records, each specifying a particular metal potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
key	a4	potential key. See table 4.15
variable 1	real	potential parameter see table 4.15

variable 2	real	potential parameter see table 4.15
variable 3	real	potential parameter see table 4.15
variable 4	real	potential parameter see table 4.15
variable 5	real	potential parameter see table 4.15
variable 6	real	potential parameter see table 4.15
variable 7	real	potential parameter see table 4.15

The variables pertaining to each potential are described in table 4.15. Note that any metal potential not specified in the FIELD file, will be assumed to be zero. This includes cross terms for alloys!

Table 4.15: Metal Potential

key	potential type	Variables (1-7)							functional form
eam	EAM								tabulated potential
fnsc	Finnis-Sinclair	c_0	c_1	c_2	c	A	d	β	$U_i(r) = \frac{1}{2}\sum_{j\neq i}(r_{ij}-c)^2(c_0+c_1 r_{ij}+c_2 r_{ij}^2) - A\sqrt{\rho_i}$ $\rho_i = \sum_{j\neq i}\left[(r_{ij}-d)^2 + \beta\frac{(r_{ij}-d)^3}{d}\right]$
stch	Sutton-Chen	ϵ	a	n	m	c			$U_i(r) = \epsilon\left[\frac{1}{2}\sum_{j\neq i}\left(\frac{a}{r_{ij}}\right)^n - c\sqrt{\rho_i}\right]$ $\rho_i = \sum_{j\neq i}\left(\frac{a}{r_{ij}}\right)^m$
gupt	Gupta	A	r_0	p	B	q_{ij}			$U_i(r) = \frac{1}{2}\sum_{j\neq i} A\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right) - B\sqrt{\rho_i}$ $\rho_i = \sum_{j\neq i}\exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right)$

Both EAM and FSM potentials can handle alloys, but care must be taken to enter the cross terms of the potentials explicitly. **Note** that the rules for defining cross terms of the potential are not the usual rules encountered in Lennard-Jones systems (see section 2.3.5).

4.1.3.5 The Tersoff Potential

The Tersoff potential [5] is designed to reproduce the effects of covalency in systems composed of group 4 elements in the periodic table (carbon, silicon, germanium etc) and their alloys. Like the metal potentials these are also non-bonded potentials characterised by atom types rather than specific atomic indices. The input of Tersoff potential data is signalled by the directive:

tersoff *n*

Where n is the number of specified Tersoff potentials. It is followed by $2n$ records specifying n particular Tersoff single atom type parameters and $n(n+1)/2$ records specifying cross atom type parameters in the following manner:

potential 1 : record 1		
atmnam	a8	atom type
key	a4	potential key, see Table 4.16
variable 1	real	potential parameter, see Table 4.16
variable 2	real	potential parameter, see Table 4.16
variable 3	real	potential parameter, see Table 4.16
variable 4	real	potential parameter, see Table 4.16
variable 5	real	cutoff range for this potential (Å) 4.16
potential 1 : record 2		
variable 6	real	potential parameter, see Table 4.16
variable 7	real	potential parameter, see Table 4.16
variable 8	real	potential parameter, see Table 4.16
variable 9	real	potential parameter, see Table 4.16
variable 10	real	potential parameter, see Table 4.16
variable 11	real	potential parameter, see Table 4.16
...	...	...
...	...	...
potential n **: record** $2n-1$		
...	...	...
potential n **: record** $2n$		
...	...	...
cross term 1 **: record** $2n+1$		
atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
variable a	real	potential parameter, see Table 4.16
variable b	real	potential parameter, see Table 4.16
...	...	...
...	...	...
cross term $n(n+1)/2$ **: record** $2n+n(n+1)/2$		
...	...	...

The variables pertaining to each potential are described in Table 4.16.

Note that the 11 parameters A to h required for the cross interactions between dissimilar elements are calculated internally by DL_POLY Classic using the prescription given by Tersoff [5]. There is no prescription for the χ and ω cross parameters, so these must be given explicitly.

Note also that the fifth variable is the range at which the particular Tersoff potential is truncated. The distance is in Å.

Table 4.16: Tersoff Potential

key	potential type	Variables (1-5,6-11,a-b)						functional form
ters	Tersoff	A	a	B	b	R		Potential form
	(single)	S	β	η	c	d	h	as shown in Section
	(cross)	χ	ω					2.3.3

4.1.3.6 External Field

The presence of an external field is flagged by the **extern** directive. The next line in the FIELD file should have another directive indicating what type of field is to be applied. On the following lines comes the `mxfld` parameters, five per line, that describe the field. In the include files supplied with DL_POLY Classic `mxfld` is set to 10.
The variables pertaining to each potential are described in table 4.17.

Table 4.17: External fields

key	potential type	Variables (1-5)					functional form†
elec	Electric field	E_x	E_y	E_z			$\underline{F} = q.\underline{E}$
oshm	Oscillating Shear	A	n				$\underline{F}_x = Acos(2n\pi.z/L_z)$
shrx	Continuous Shear	A	z_0				$\mid z \mid > z_0$: $\underline{v}_x = (1/2)A(\mid z \mid /z)$
grav	Gravitational Field	G_x	G_y	G_z			$\underline{F} = m.\underline{G}$
magn	Magnetic Field	H_x	H_y	H_z			$\underline{F} = q(\underline{v} \times \underline{H})$
sphr	Containing Sphere	A	R_0	n	$R_{\rm cut}$		$r > R_{\rm cut}$: $\underline{F} = A(R_0 - r)^{-n}$
zbnd	Repulsive wall (harmonic)	A	Z_0	$f = \pm 1$			$zf > Z_0 f$: $F_z = -A(z - Z_0)$
zres	Restraint zone (harmonic)	n_1	n_2	A	z_{min}	z_{max}	$\underline{F} = A(z_{max} - z_{com})$: $z_{com} > z_{max}$ $\underline{F} = A(z_{min} - z_{com})$: $z_{com} < z_{min}$ $z_{com} = \sum_{i=n_1}^{n_2} m_i z_i / M$ $M = \sum_{i=n_1}^{n_2} m_i$

4.1.3.7 Closing the FIELD File

The FIELD file must be closed with the directive:

close

which signals the end of the force field data. Without this directive DL_POLY Classic will abort.

4.1.4 The REVOLD File

This file contains statistics arrays from a previous job. It is not required if the current job is not a continuation of a previous run (ie. if the **restart** directive is not present in the CONTROL file - see above). The file is unformatted and therefore not readable by normal people. DL_POLY Classic normally produces the file REVIVE (see section 4.2.5) at the end of a job which contains the statistics data. REVIVE should be copied to REVOLD before a continuation run commences. This may be done by the *copy* macro supplied in the *execute* sub-directory of DL_POLY Classic.

4.1.4.1 Format

The REVOLD file is unformatted. All variables appearing are written in native real*8 representation. Nominally integer quantities (e.g. the timestep number `nstep`) are represented by the the nearest real number. The contents are as follows (the dimensions of array variables are given in brackets and are defined in the appropriate Fortran modules).

record 1:

`nstep`	timestep of final configuration
`numacc`	number of configurations used in averages
`numrdf`	number of configurations used in rdf averages
`chit`	thermostat momentum
`chip`	barostat momentum
`conint`	conserved quantity for selected ensemble
`nzden`	number of configurations used in z density

record 2:

`virtot`	total system virial
`vircom`	rigid body COM virial
`eta`	scaling factors for simulation cell matrix elements (9)
`strcns`	constraint stress tensor elements (9)
`strbod`	rigid body stress tensor elements (9)

record 3:

`stpval`	instantaneous values of thermodynamic variables (`mxnstk`)

record 4:

`sumval`	average values of thermodynamic variables (`mxnstk`)

record 5:

`ssqval`	fluctuation (squared) of thermodynamic variables (`mxnstk`)

record 6:

`zumval`	running totals of thermodynamic variables (`mxnstk`)

record 7:

`ravval`	rolling averages of thermodynamic variables (`mxnstk`)

record 8:

`stkval`	stacked values of thermodynamic variables (`mxstak`×`mxnstk`)

record 9:

`xx0`	x component of atomic displacement (MSD) (`mxatms`)
`yy0`	y component of atomic displacement (MSD) (`mxatms`)
`zz0`	z component of atomic displacement (MSD) (`mxatms`)

record 10:

`xxs`	x-coordinates of tether points (`mxatms`)
`yys`	y-coordinates of tether points (`mxatms`)
`zzs`	z-coordinates of tether points (`mxatms`)

record 11:

`rdf`	(Optional) RDF array (`mxrdf`×`mxvdw`)

record 12:

`zdens`	(Optional) z-density array (`mxrdf`×`mxsvdw`)

4.1.4.2 Further Comments

Note that recompiling DL_POLY Classic with a different DL_PARAMS.INC file, may render any existing REVOLD file unreadable by the code.

4.1.5 The TABLE File

The TABLE file provides an alternative way of reading in the short range potentials - in tabular form. This is particularly useful if an analytical form of the potential does not exist or is too complicated to specify in the FORGEN subroutine. The table file is read by the subroutine FORTAB.F in the VDW_TERMS.F file..

The option of using tabulated potentials is specified in the FIELD file (see above). The specific potentials that are to be tabulated are indicated by the use of the **tab** keyword on the record defining the short range potential (see table 4.12). The directive **vdwtable** may be used in place of **vdw** to indicate that one or more of the short ranged potentials is specified in the form of a table.

4.1.5.1 Format

The file is fixed-formatted with integers as "i10", reals as "e15.8". Character variables are read as "a8". The header record is formatted as 80 alphanumeric characters.

4.1.5.2 Definitions of Variables

record 1		
`header`	a80	file header
record 2		
`delpot`	real	mesh resolution in Å
`cutpot`	real	cutoff used to define tables Å
`ngrid`	integer	number of grid points in tables

The subsequent records define each tabulated potential in turn, in the order indicated by the specification in the FIELD file. Each potential is defined by a header record and a set of data records with the potential and force tables.

header record:		
`atom 1`	a8	first atom type
`atom 2`	a8	second atom type
potential data records:	($number\ of\ data\ records = Int((\texttt{ngrid}+3)/4)$)	
`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4
force data records:	($number\ of\ data\ records = Int((\texttt{ngrid}+3)/4)$)	
`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4

4.1.5.3 Further Comments

It should be noted that the number of grid points in the TABLE file should not be less than the number of grid points DL_POLY Classic is expecting. (This number is given by the parameter `mxgrid`,

which is defined in the PARSET.F subroutine in the SETUP_PROGRAM.F file.) DL_POLY Classic will re-interpolate the tables if `ngrid`≥`mxgrid`, but will abort if `ngrid`<`mxgrid`.

The potential and force tables are used to fill the internal arrays `vvv` and `ggg` respectively (see section 2.3.1). The contents of force arrays are derived from the potential via the formula:

$$G(r) = -r\frac{\partial}{\partial r}U(r).$$

Note this is a virial expression and *not* the same as the true force.

Important The potential and force arrays in the TABLE file are written in the same units as the FIELD file. So if you specified a particular unit using the **UNITS** directive in the FIELD file, the same units are expected here. It is useful to note that the definition of the force arrays given above means that the units are the same as for the potential - i.e. are handled using the same conversion factors.

4.1.6 The TABEAM File

The TABEAM file contains the tabulated potential functions (no explicit analytic form) describing the metal interactions in the MD system. This file is read by the subroutine METTAB.

The EAM potential for an n component metal alloy requires the specification of n electron density functions (one for each atom type) and n embedding functions (again one for each atom type) and $n(n+1)/2$ *cross* pair potential functions. This makes $n(n+5)/2$ functions in total. **Note** that the option of using EAM interactions must also be *explicitly* declared in the FIELD file so that for the n component alloy there are $n(n+1)/2$ *cross* pair potential (**eam**) keyword entries in FIELD (see above). (**Note** that all metal interactions must be of the same type!)

4.1.6.1 The TABEAM File Format

The file is free-formatted but blank and commented lines are not allowed.

4.1.6.2 Definitions of Variables

record 1		
`header`	a100	file header
record 2		
`numpot`	integer	number of potential functions in file

The subsequent records define the $n(n+5)/2$ functions for an n component alloy - n electron density functions (one for each atom type) - **dens**ity keyword, n embedding functions (again one for each atom type) - **embe**ding keyword, and $n(n+1)/2$ *cross* pair potential functions - **pair**s keyword. The functions may appear in any random order in TABEAM as their identification is based on their unique keyword, defined first in the function's header record. The header record is followed by a predefined number of data records **as a maximum of four data per record are read in** - allowing for incompletion of the very last record.

header record:		
keyword	a4	type of EAM function: **dens**, **embe**d or **pair**
`atom 1`	a8	first atom type
`atom 2`	a8	second atom type - only specified for **pair** potential functions
`ngrid`	integer	number of function data points to read in
`limit 1`	real	lower interpolation limit in Åfor **dens** and **pair**

		or in density units for **embe**d
`limit 2`	real	upper interpolation limit in Åfor **dens** and **pair**
		or in density units for **embe**d

function data records: (*number of data records* = `Int`((`ngrid`+3)/4))

`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4

4.2 The OUTPUT Files

DL_POLY Classic produces up to eight output files: HISTORY, OUTPUT, REVCON, REVIVE, RDFDAT, ZDNDAT, STATIS and CFGMIN. These respectively contain: a dump file of atomic coordinates, velocities and forces; a summary of the simulation; the restart configuration; statistics accumulators; radial distribution data, Z-density data, a statistical history, and the configuration with the lowest configurational energy. Some of these files are optional and appear only when certain options are used.

Note: In addition to the files described in this chapter, users of the hyperdynamics features of DL_POLY Classic should see Chapter 5, where additional files specific to that purpose are described. Similarly, the output files specific to the solvation features of DL_POLY Classic are descrived in Chapter 6.

4.2.1 The HISTORY File

The HISTORY file is the dump file of atomic coordinates, velocities and forces. Its principal use is for off-line analysis. The file is written by the subroutines TRAJECT or TRAJECT_U. The control variables for this file are `ltraj, nstraj, istraj` and `keytrj` which are created internally, based on information read from the **traj** directive in the CONTROL file (see above). The HISTORY file will be created only if the directive **traj** appears in the CONTROL file. Note that the HISTORY file can be written in either a formatted or unformatted version. We describe each of these separately below. If you want your HISTORY data to have maximum numerical precision, you should use the unformatted version.

The HISTORY file can become *very* large, especially if it is formatted. For serious simulation work it is recommended that the file be written to a scratch disk capable of accommodating a large data file. Alternatively the file may be written as unformatted (below), which has the additional advantage of speed. However, writing an unformatted file has the disadvantage that the file may not be readily readable except by the machine on which it was created. This is particularly important if graphical processing of the data is required.

4.2.1.1 The Formatted HISTORY File

The formatted HISTORY file is written by the subroutine TRAJECT and has the following structure.

record 1 (a80)

`header`	a80	file header

record 2 (3i10)

`keytrj`	integer	trajectory key (see table 4.3)
`imcon`	integer	periodic boundary key (see table 4.6)
`natms`	integer	number of atoms in simulation cell

For timesteps greater than `nstraj` the HISTORY file is appended at intervals specified by the **traj** directive in the CONTROL file, with the following information for each configuration:

record i (a8,4i10,f12.6)

`timestep`	a8	the character string “timestep”
`nstep`	integer	the current time-step
`natms`	integer	number of atoms in configuration
`keytrj`	integer	trajectory key (again)
`imcon`	integer	periodic boundary key (again)

`tstep`	real	integration timestep
record ii (3g12.4) for `imcon` > 0		
`cell(1)`	real	x component of a cell vector
`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
record iii (3g12.4) for `imcon` > 0		
`cell(4)`	real	x component of b cell vector
`cell(5)`	real	y component of b cell vector
`cell(6)`	real	z component of b cell vector
record iv (3g12.4) for `imcon` > 0		
`cell(7)`	real	x component of c cell vector
`cell(8)`	real	y component of c cell vector
`cell(9)`	real	z component of c cell vector

This is followed by the configuration for the current timestep. i.e. for each atom in the system the following data are included:

record a (a8,i10,2f12.6)		
`atmnam`	a8	atomic label
`iatm`	i10	atom index
`weight`	f12.6	atomic mass (a.m.u.)
`charge`	f12.6	atomic charge (e)
record b (3e12.4)		
`xxx`	real	x coordinate
`yyy`	real	y coordinate
`zzz`	real	z coordinate
record c (3e12.4) only for `keytrj` > 0		
`vxx`	real	x component of velocity
`vyy`	real	y component of velocity
`vzz`	real	z component of velocity
record d (3e12.4) only for `keytrj` > 1		
`fxx`	real	x component of force
`fyy`	real	y component of force
`fzz`	real	z component of force

Thus the data for each atom is a minimum of two records and a maximum of 4.

4.2.1.2 The Unformatted HISTORY File

The unformatted HISTORY file is written by the subroutine TRAJECT_U and has the following structure:

record 1	
`header`	configuration name (character*80)
record 2	
`natms`	number of atoms in the configuration (real*8)
record 3	
`atname(1,...,natms)`	atom names or symbols (character*8)
record 4	

`weight(1,...,natms)`	atomic masses (real*8)
record 5	
`charge(1,...,natms)`	atomic charges (real*8)

For time-steps greater than `nstraj`, the HISTORY file is appended, at intervals specified by the **traj** directive in the CONTROL file, with the following information:

record i	
`nstep`	the current time-step (real*8)
`natms`	number of atoms in configuration (real*8)
`keytrj`	trajectory key (real*8)
`imcon`	image convention key (real*8)
`tstep`	integration timestep (real*8)
record ii for `imcon` > 0	
`cell(1,...,9)`	a, b and c cell vectors (real*8)
record iii	
`xxx(1,...,natms)`	atomic x-coordinates (real*8)
record iv	
`yyy(1,...,natms)`	atomic y-coordinates (real*8)
record v	
`zzz(1,...,natms)`	atomic z-coordinates (real*8)
record vi only for `keytrj`> 0	
`vxx(1,...,natms)`	atomic velocities x-component (real*8)
record vii only for `keytrj`> 0	
`vyy(1,...,natms)`	atomic velocities y-component (real*8)
record viii only for `keytrj`> 0	
`vzz(1,...,natms)`	atomic velocities z-component (real*8)
record ix only for `keytrj`> 1	
`fxx(1,...,natms)`	atomic forces x-component (real*8)
record x only for `keytrj`> 1	
`fyy(1,...,natms)`	atomic forces y-component (real*8)
record xi only for `keytrj`> 1	
`fzz(1,...,natms)`	atomic forces z-component (real*8)

Note the implied conversion of integer variables to real on record i.

4.2.2 The OUTPUT File

The job output consists of 7 sections: Header; Simulation control specifications; Force field specification; Summary of the initial configuration; Simulation progress; Summary of statistical data; Sample of the final configuration; and Radial distribution functions. These sections are written by different subroutines at various stages of a job. Creation of the OUTPUT file *always* results from running DL_POLY Classic. It is meant to be a human readable file, destined for hardcopy output.

4.2.2.1 Header

Gives the DL_POLY Classic version number, the number of processors used and a title for the job as given in the header line of the input file CONTROL. This part of the file is written from the subroutines DLPOLY and SIMDEF

4.2.2.2 Simulation Control Specifications

Echoes the input from the CONTROL file. Some variables may be reset if illegal values were specified in the CONTROL file. This part of the file is written from the subroutine SIMDEF.

4.2.2.3 Force Field Specification

Echoes the FIELD file. A warning line will be printed if the system is not electrically neutral. This warning will appear immediately before the non-bonded short-range potential specifications. This part of the file is written from the subroutine SYSDEF.

4.2.2.4 Summary of the Initial Configuration

This part of the file is written from the subroutine SYSGEN. It states the periodic boundary specification, the cell vectors and volume (if appropriate) and the initial configuration of (a maximum of) 20 atoms in the system. The configuration information given is based on the value of `levcfg` in the CONFIG file. If `levcfg` is 0 (or 1) positions (and velocities) of the 20 atoms are listed. If `levcfg` is 2 forces are also written out.

For periodic systems this is followed by the long range corrections to the energy and pressure.

4.2.2.5 Simulation Progress

This part of the file is written by the DL_POLY Classic root segment DLPOLY. The header line is printed at the top of each page as:

```
-----------------------------------------------------------------------------------------------------

   step   eng_tot  temp_tot   eng_cfg   eng_vdw   eng_cou   eng_bnd   eng_ang   eng_dih   eng_tet
time       eng_pv  temp_rot   vir_cfg   vir_vdw   vir_cou   vir_bnd   vir_ang   vir_con   vir_tet
cpu time   volume  temp_shl   eng_shl   vir_shl     alpha      beta   gamma   vir_pmf     press

-----------------------------------------------------------------------------------------------------
```

The labels refer to :

line 1

step	MD step number
`eng_tot`	total internal energy of the system
`temp_tot`	system temperature
`eng_cfg`	configurational energy of the system
`eng_vdw`	configurational energy due to short-range potentials
`eng_cou`	configurational energy due to electrostatic potential
`eng_bnd`	configurational energy due to chemical bond potentials
`eng_ang`	configurational energy due to valence angle and three-body potentials
`eng_dih`	configurational energy due to dihedral inversion and four-body potentials
`eng_tet`	configurational energy due to tethering potentials

line 2

`time`	elapsed simulation time (fs,ps,ns) since the beginning of the job
`eng_pv`	enthalpy of system
`temp_rot`	rotational temperature
`vir_cfg`	total configurational contribution to the virial
`vir_vdw`	short range potential contribution to the virial
`vir_cou`	electrostatic potential contribution to the virial

`vir_bnd`	chemical bond contribution to the virial
`vir_ang`	angular and three body potentials contribution to the virial
`vir_con`	constraint bond contribution to the virial
`vir_tet`	tethering potential contribution to the virial
line 3	
`cpu time`	elapsed cpu time (s,m,h,d) since the beginning of the job
`volume`	system volume
`temp_shl`	core-shell temperature
`eng_shl`	configurational energy due to core-shell potentials
`vir_shl`	core-shell potential contribution to the virial
`alpha`	angle between b and c cell vectors
`beta`	angle between c and a cell vectors
`gamma`	angle between a and b cell vectors
`vir_pmf`	Potential of mean force constraint contribution to the virial
`press`	pressure

Note: The total internal energy of the system (variable `tot_energy`) includes all contributions to the energy (including system extensions due to thermostats etc.) It is nominally the *conserved variable* of the system, and is not to be confused with conventional system energy, which is a sum of the kinetic and configuration energies.

The interval for printing out these data is determined by the directive **print** in the CONTROL file. At each time-step that printout is requested the instantaneous values of the above statistical variables are given in the appropriate columns. Immediately below these three lines of output the rolling averages of the same variables are also given. The maximum number of time-steps used to calculate the rolling averages is determined by the parameter `mxstak` defined in the SETUP_MODULE./F file. The working number of time-steps for rolling averages is controlled by the directive **stack** in file CONTROL (see above). The default value is `mxstak`.

Energy Units: The energy unit for the data appearing in the OUTPUT is defined by the **units** directive appearing in the CONTROL file.

Pressure units: The unit of pressure is k *atm*, irrespective of what energy unit is chosen.

4.2.2.6 Summary of Statistical Data

This portion of the OUTPUT file is written from the subroutine RESULT. The number of time-steps used in the collection of statistics is given. Then the averages over the production portion of the run are given for the variables described in the previous section. The root mean square variation in these variables follow on the next two lines. The energy and pressure units are as for the preceeding section.

Also provided in this section is an estimate of the diffusion coefficient for the different species in the simulation, which is determined from a *single time origin* and is therefore very approximate. Accurate determinations of the diffusion coefficients can be obtained using the MSD utility program, which processes the HISTORY file (see chapter 10).

If an NPT or N$\underline{\underline{\sigma}}$T simulation is performed the OUTPUT file also provides the mean stress (pressure) tensor and mean simulation cell vectors.

4.2.2.7 Sample of Final Configuration

The positions, velocities and forces of the 20 atoms used for the sample of the initial configuration (see above) are given. This is written by the subroutine RESULT.

4.2.2.8 Radial Distribution Functions

If both calculation and printing of radial distribution functions have been requested (by selecting directives **rdf** and **print rdf** in the CONTROL file) radial distribution functions are printed out. This is written from the subroutine RDF1. First the number of time-steps used for the collection of the histograms is stated. Then each function is given in turn. For each function a header line states the atom types ('a' and 'b') represented by the function. Then r, $g(r)$ and $n(r)$ are given in tabular form. Output is given from 2 entries before the first non-zero entry in the $g(r)$ histogram. $n(r)$ is the average number of atoms of type 'b' within a sphere of radius r around an atom of type 'a'. Note that a readable version of these data is provided by the RDFDAT file (below).

4.2.2.9 Z Density Profile

If both calculation and printing of Z density profiles has been requested (by selecting directives **zden** and **print rdf** in the CONTROL file Z density profiles are printed out as the last part of the OUTPUT file. This is written by the subroutine ZDEN1. First the number of time-steps used for the collection of the histograms is stated. Then each function is given in turn. For each function a header line states the atom type represented by the function. Then z, $\rho(z)$ and $n(z)$ are given in tabular form. Output is given from $Z = [-L/2, L/2]$ where L is the length of the MD cell in the Z direction and $\rho(z)$ is the mean number density. $n(z)$ is the running integral from $-L/2$ to z of (xy cell area)$\rho(s)ds$. Note that a readable version of these data is provided by the ZDNDAT file (below).

4.2.3 The REVCON File

This file is formatted and written by the subroutine REVIVE. REVCON is the restart configuration file. The file is written every `ndump` time steps in case of a system crash during execution and at the termination of the job. A successful run of DL_POLY Classic will always produce a REVCON file, but a failed job may not produce the file if an insufficient number of timesteps have elapsed. `ndump` is a parameter defined in the SETUP_MODULE.F file found in the *source* directory of DL_POLY Classic. Changing `ndump` necessitates recompiling DL_POLY Classic.

REVCON is identical in format to the CONFIG input file (see section 4.1.2).

REVCON should be renamed CONFIG to continue a simulation from one job to the next. This is done for you by the *copy* macro supplied in the *execute* directory of DL_POLY Classic.

4.2.4 The CFGMIN File

The CFGMIN file only appears if the user has selected the programmed minimisation option (directive **minim** in the CONTROL file). Its contents have the same format as the CONFIG file (see section 4.1.2), but contains only atomic position data and will never contain either velocity or force data (i.e. parameter `levcfg` is always zero). In addition, two extra numbers appear on the end of the second line of the file:

1. an integer indicating the number of minimisation cycles required to obtain the structure (format I10);

2. the configuration energy of the final structure expressed in DL_POLY units 1.3.10 (format F20).

4.2.5 The REVIVE File

This file is unformatted and written by the subroutine REVIVE. It contains the accumulated statistical data. It is updated whenever the file REVCON is updated (see previous section). REVIVE should be renamed REVOLD to continue a simulation from one job to the next. This is done by the *copy* macro supplied in the *execute* directory of DL_POLY Classic. In addition, to continue a simulation from a previous job the **restart** keyword must be included in the CONTROL file.

The format of the REVIVE file is identical to the REVOLD file described in section 4.1.4.

4.2.6 The RDFDAT File

This is a formatted file containing em Radial Distribution Function (RDF) data. Its contents are as follows:

record 1		
`cfgname`	character (A80)	configuration name
record 2		
`ntpvdw`	integer (i10)	number of RDFs in file
`mxrdf`	integer (i10)	number of data points in each RDF

There follow the data for each individual RDF i.e. *ntpvdw* times. The data supplied are as follows:

first record		
`atname 1`	character (A8)	first atom name
`atname 2`	character (A8)	second atom name
following records (*mxrdf* records)		
`radius`	real (e14)	interatomic distance (A)
`g(r)`	real (e14)	RDF at given radius.

Note the RDFDAT file is optional and appears when the **print rdf** option is specified in the CONTROL file.

4.2.7 The ZDNDAT File

This is a formatted file containing the Z-density data. Its contents are as follows:

record 1		
`cfgname`	character (A80)	configuration name
record 2		
`mxrdf`	integer (i10)	number of data points in the Z-density function
following records (*mxrdf* records)		
`z`	real (e14)	distance in z direction (A)
$\rho(z)$	real (e14)	Z-density at given height `z`

Note the ZDNDAT file is optional and appears when the **print rdf** option is specified in the CONTROL file.

4.2.8 The STATIS File

The file is formatted, with integers as "i10" and reals as "e14.6". It is written by the subroutine STATIC. It consists of two header records followed by many data records of statistical data.

record 1

`cfgname`	character	configuration name

record 2

`string`	character	energy units

Data records

Subsequent lines contain the instantaneous values of statistical variables dumped from the array `stpval`. A specified number of entries of `stpval` are written in the format "(1p,5e14.6)". The number of array elements required (determined by the parameter `mxnstk` in the DL_PARAMS.INC file) is

$$\begin{aligned} \texttt{mxnstk} \geq \quad & 27 + \texttt{ntpatm}(\text{number of unique atomic sites}) \\ & +9(\text{if stress tensor calculated}) \\ & +9(\text{if constant pressure simulation requested}) \end{aligned}$$

The STATIS file is appended at intervals determined by the **stats** directive in the CONTROL file. The energy unit is as specified in the CONTROL file with the the **units** directive, and are compatible with the data appearing in the OUTPUT file. The contents of the appended information is:

record i

`nstep`	integer	current MD time-step
`time`	real	elapsed simulation time = `nstep`$\times\Delta t$
`nument`	integer	number of array elements to follow

record ii `stpval`(1) – `stpval`(5)

`engcns`	real	total extended system energy (i.e. the conserved quantity)
`temp`	real	system temperature
`engcfg`	real	configurational energy
`engsrp`	real	VdW/metal/Tersoff energy
`engcpe`	real	electrostatic energy

record iii `stpval`(6) – `stpval`(10)

`engbnd`	real	chemical bond energy
`engang`	real	valence angle/3-body potential energy
`engdih`	real	dihedral/inversion/four body energy
`engtet`	real	tethering energy
`enthal`	real	enthalpy (total energy + PV)

record iv `stpval`(11) – `stpval`(15)

`tmprot`	real	rotational temperature
`vir`	real	total virial
`virsrp`	real	VdW/metal/Tersoff virial
`vircpe`	real	electrostatic virial
`virbnd`	real	bond virial

record v `stpval`(16) -`stpval`(20)

`virang`	real	valence angle/3-body virial

`vircon`	real	constraint virial
`virtet`	real	tethering virial
`volume`	real	volume
`tmpshl`	real	core-shell temperature
record vi `stpval`(21) `-stpval`(25)		
`engshl`	real	core-shell potential energy
`virshl`	real	core-shell virial
`alpha`	real	MD cell angle α
`beta`	real	MD cell angle β
`gamma`	real	MD cell angle γ
record vii `stpval`(26) `-stpval`(27)		
`virpmf`	real	Potential of Mean Force virial
`press`	real	pressure
the next `ntpatm` **entries**		
`amsd(1)`	real	mean squared displacement of first atom types
`amsd(2)`	real	mean squared displacement of second atom types
`...`	...	...
`amsd(ntpatm)`	real	mean squared displacement of last atom types
the next 9 entries - ***if*** **the stress tensor is calculated**		
`stress(1)`	real	xx component of stress tensor
`stress(2)`	real	xy component of stress tensor
`stress(3)`	real	xz component of stress tensor
`stress(4)`	real	yx component of stress tensor
`...`	real	...
`stress(9)`	real	zz component of stress tensor
the next 9 entries - ***if*** **a NPT simulation is undertaken**		
`cell(1)`	real	x component of a cell vector
`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
`cell(4)`	real	x component of b cell vector
`...`	real	...
`cell(9)`	real	z component of c cell vector

Chapter 5

Hyperdynamics

Scope of Chapter

This chapter describes the facilties within DL_POLY Classic for performing accelerated dynamics (or hyperdynamics) using the *Bias Potential Dynamics* and *Temperature Accelerated Dynamics* methods.

5.1 Overview of Hyperdynamics

The first thing to note about the hyperdynamics methods in DL_POLY Classic is that they were designed for studies of kinetic processes in the solid state, which mostly means diffusion. In solids diffusion is characterised by infrequent atomic 'hops' occurring on a time scale of order 100 ps to 1000 ps per hop, which is too infrequent to give a measurable diffusion in a normal molecular dynamics simulation. Hyperdynamics methods are designed to overcome this problem by accelerating the hopping frequency.

The hyperdynamics methods built into DL_POLY Classic are *Bias Potential Dynamics* (BPD) [62] and *Temperature Accelerated Dynamics* (TAD) [63], both of which were conceived by Voter *et al*, though the implementation of BPD in the program uses the bias potential devised by Hamelberg, Mongan and McCammon [64], which is simpler to use. In passing it is useful to note that BPD can be used to improve configurational sampling in systems other than solids and this facility has been retained in the DL_POLY Classic implementation (see section 5.3.5).

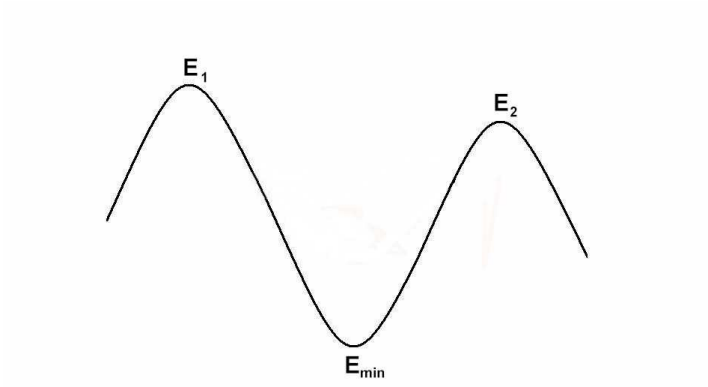

Figure 5.1: Model Potential Energy Surface.
The potential energy surface of a solid is characterised by deep energy basins, such as E_{min}, representing the various structural states. Escape to other states (i.e. diffusion) must go via 'saddle points' on the surface indicated by points E_1 and E_2. The energy differences $(E_1 - E_{min})$ or $(E_2 - E_{min})$ represent the activation energies (E^*) required to enable escape via the respective saddle points. Thermal excitation alone is insufficient to achieve escape in a reasonable time.

The basic problem in simulating diffusion in solids is that each possible structure of the system is trapped in a deep basin in the potential energy surface (see figure 5.1) representing a particular 'state'. For diffusion to occur the system must become sufficiently thermally excited to achieve the activation energy E^* necessary to escape. In dimensions higher than 1, E^* represents a 'saddle point' on the potenial energy surface. Special techniques are required to accelerate the escape and achieve a measurable diffusion in a reasonable time. These however must be devised so that the kinetic processes of the original system may be faithfully reconstructed. Both the BPD and TAD methods in DL_POLY Classic, which are respectively described in sections 5.3 and 5.4 below, satisfy this requirement.

It is apparent from the discussion above that an important requirement in hyperdynamics is the calculation of the activation energy, which is equivalent to determining the saddle point between two states. This is accomplished in DL_POLY Classic by a technique known as the 'Nudged Elastic Band' (NEB) method. Understanding the NEB method is a prerequisite for using the DL_POLY Classic hyperdynamics methods correctly, so a description of it is given in the following section.

5.2 The Nudged Elastic Band Calculation

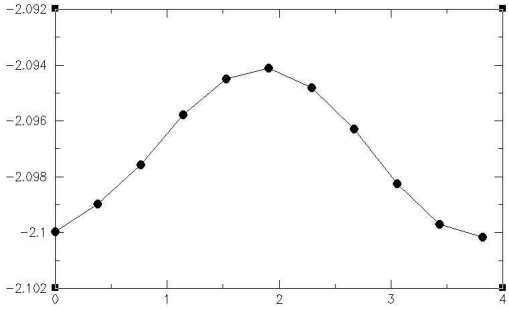

Figure 5.2: Basic NEB Theory
Plot of bead configuration energy vs. reaction path for a NEB calculation of a structural transition in a Lennard Jones solid.

The 'Nudged Elastic Band' (NEB) method is a standard method for determining the energy optimised pathway between two known structures. In DL_POLY Classic it is used to find the escape pathway (also called the 'reaction path') between structural basins, yielding the activation energy in the process. The implementation is based on the method described by Henkelman and Jonsson[65] though it has been adapted to work in parallel. The method is as follows.

1. The start and end points of the NEB construction are the energy minimised structures for states A and B. (A structure ($\underline{R}^N$) is defined as the set of $3N$ coordinates locating all N atoms in the system.)

2. A series of states is constructed by linear interpolation between the structures of states A and B i.e. a series of configurations $\underline{R}_i^N$ is generated with $i = 0, \ldots, N_{neb}$ such that $i = 0$ indicates state A and $i = N_{neb}$ indicates state B and

$$\underline{R}_i^N = \underline{R}_0^N + (i/N_{neb}) * (\underline{R}_{N_{neb}}^N - \underline{R}_0^N) \tag{5.1}$$

   For convenience these configurations are called the 'beads' of the NEB 'chain'. Each bead has a configuration energy which may be written as $V_c(\underline{R}_i^N)$. This is the usual configuration energy for a system with an atomic structure $\underline{R}_i^N$.

3. Each bead in the NEB chain is then connected to its two nearest neighbours by a harmonic spring (except for the end beads which have only one neighbour each), so that the beads make a chain strung from state A to state B. The spring energy of the whole chain is then defined as

$$V_s(\underline{R}^N_{N_{neb}}) = \frac{1}{2} k_{neb} \sum_{i=1}^{N_{neb}} (\underline{R}^N_i - \underline{R}^N_{i-1})^2 \tag{5.2}$$

where k_{neb} is the spring force constant.

4. With the chain thus defined, the objective is now to minimise the energy function $E(\underline{R}^N_{N_{neb}})$ where

$$E(\underline{R}^N_{N_{neb}}) = V_s(\underline{R}^N_{N_{neb}}) + \sum_{i=1}^{N_{neb}-1} V_c(\underline{R}^N_i) \tag{5.3}$$

in which the adjustable variables are the configurations $\underline{R}^N_i$ (i.e. the atomic coordinates in each structure), while the chain end beads at $\underline{R}^N_0$ and $\underline{R}^N_{N_{neb}}$ remain fixed.

5. It is clear that the unconstrained configurations $\underline{R}^N_i$ would normally relax into the nearest local minimum, but that this cannot happen if they are sufficiently constrained by the harmonic springs (i.e. k_{neb} is strong enough). Thus the minimisation of the chain will tend to locate each bead in a position along a path between states A and B like a stretched necklace, which approximates the minimum energy path between the two states.

6. In practice this simple idea needs refining (or 'nudging'). Thus care is taken to ensure that the springs forces acting on the beads and the forces optimising bead configurations are approximately orthogonal. This means that the atomic forces are zeroed in directions parallel to the path of the chain and the spring forces are zeroed in directions normal to the chain. The method of Henkelman and Jonsson [65] is designed to achieve this.

7. If the NEB optimisation works correctly, the result will be that beads are evenly spaced along the minimum energy path (see figure 5.2). Then by fitting the energies of the beads as a function of the distance along the path, the maximum energy (i.e. E^*) along the path may be obtained. The DL_POLY Classic NEB routine does this fit using third order splines.

5.3 Bias Potential Dynamics

5.3.1 Theory of Bias Potential Dynamics

BPD works on the simple principle that the addition of a suitable potential term to original system potential can have the effect of reducing the depth of the potential basin (see figure 5.3) so assisting escape to neighbouring states. The biased system potential ($V_{bias}(\underline{R}^N)$) is thus given by

$$V_{bias}(\underline{R}^N) = V(\underline{R}^N) + W_{bias}(\underline{R}^N) \tag{5.4}$$

where $V(\underline{R}^N)$ is the original system potential and $W_{bias}(\underline{R}^N)$ is the bias potential. Voter [62] has shown that using a bias potential accelerates the diffusion rate constant k^{TST} (as defined by Transition State Theory) by a boost factor:

$$k^{TST}_{bias} = \left\langle e^{\beta W_{bias}(\underline{R}^N)} \right\rangle_{bias} k^{TST} \tag{5.5}$$

where $\beta = 1/k_B T$ and the ensemble average, which is calculated in the biased system, represents the boost factor.

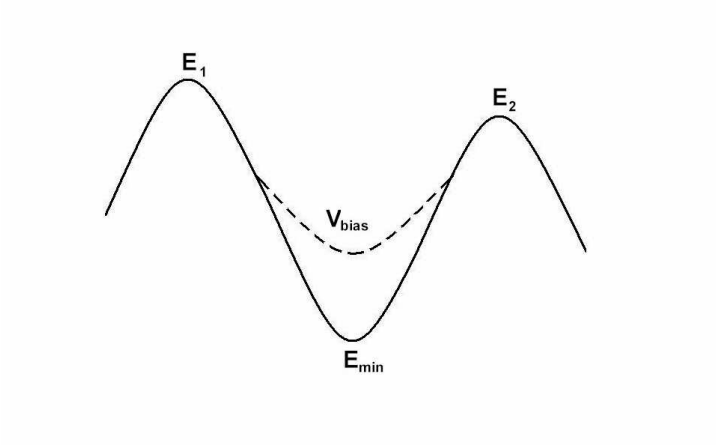

Figure 5.3: Basic BPD Theory
The normal potential energy surface (continuous line) is characterised by deep basins such as E_{min} from which escape is improbable. The biased potential V_{bias} (dashed line) reduces the basin depth, making transitions more likely. To preserve the kinetic pathway of the original system, the bias potential must be less than the saddle points E_1 and E_2 and for molecular dynamics purposes ideally should join continuously to the normal system potential (see text).

However this simple accelerative factor alone is not sufficient if a faithful description of the diffusion path in the original system is required. Voter [62] showed that to recover the true diffusional path it is important that the bias potential does not affect the structure of the transition state (i.e. the saddle points for the system potential energy surface). If this is the case then the *relative* rate constants for escape from a given structure (or state) to any other neighbouring state remains constant i.e. for transitions from state A to state B or state C:

$$\frac{k^{TST}_{A_b \to B}}{k^{TST}_{A_b \to C}} = \frac{k^{TST}_{A \to B}}{k^{TST}_{A \to C}} \tag{5.6}$$

where A_b represents state A simulated with the bias potential present. With this condition satisfied for all possible transitions a simulation will reproduce the diffusional path obtained in the original system, but at an accelerated rate.

An early difficulty with BPD was defining the bias potential. However a particularly convenient form has been devised by Hamelberg *et al* [64] which has the form

$$W_{bias}(\underline{R}^N) = H(E_{bias} - V(\underline{R}^N)) \frac{[E_{bias} - V(\underline{R}^N)]^2}{[\alpha + E_{bias} - V(\underline{R}^N)]}, \tag{5.7}$$

where α is a constant that controls the curvature of the bias potential (see below). E_{bias} is a fixed potential energy level above which the bias potential $W_{bias}(\underline{R}^N))$ becomes zero (and the unbiased potential is restored). This is controlled by $H(x)$, a Heaviside function, which is zero if the argument $x < 0$ and 1 if $x > 0$. Thus setting E_{bias} correctly provides a means to preserve the structure of the saddle points of the original surface. (Note however that the user must determine a safe value

for this.) A value of E_{bias} set above the value of the activation energy E^* anywhere on the surface invalidates Voter's condition (5.6).

Using the definition of the bias potential (5.7) it is easy to show that the atomic forces in the biased system are given by:

$$\underline{f}_i^{bias} = \underline{f}_i \left(\frac{\alpha}{\alpha + E_{bias} - V(\underline{R}^N)} \right)^2 \qquad E_{bias} > V(\underline{R}^N). \tag{5.8}$$

When $E_{bias} \leq V(\underline{R}^N)$ the atomic forces are the same as for the unbiased system.

The constant α in equation (5.7) plays an important role. If it is set to zero then $V_{bias}(\underline{R}^N) = E_{bias}$ i.e. the biased system potential (5.4) becomes a flat surface within the basins of the original potential energy surface. In this case BPD is equivalent to a technique known as "puddle skimming" [66], which is a viable method for Monte Carlo simulation, but has a disadvantage for molecular dynamics in that whenever $E_{bias} = V(\underline{R}^N)$ the atomic forces become discontinuous. For dynamics a nonzero value of α is therefore always to be preferred. Hamelberg *et al* provides a workable prescription for obtaining α in reference [64]. However for DL_POLY Classic a different approach is taken, which is outlined below.

The scheme employed in DL_POLY Classic makes use of the fact that the local potential energy minimum is known at any given instant by virtue of the minimisation operations that occur periodically in the course of the simulations. (These are necessary to check for any changes in structure.) This minimum (called hereafter V_0) is used to define the effective zero point of the configurational energy scale, which allows E_{bias} to be conveniently defined in terms of a temperature T_{bias}, such that

$$(E_{bias} - V_0) = \frac{3}{2} N k_B T_{bias} \tag{5.9}$$

This representation has the advantage that it can be intuitively related to the energetics of the system without prior knowledge of where the system resides on the absolute energy scale. One can, for example, experiment with gradual increases in T_{bias} until transitions occur at a reasonable rate.

In a similar way an energy V_{min}, with an associated temperature T_{min}, can also be defined with respect to V_0, such that V_{min} represents the actual minimum of the biased potential $V_{bias}(\underline{R}^N)$ (equation 5.7). It can easily be shown that the minima of both $V_{bias}(\underline{R}^N)$ and $V(\underline{R}^N)$ occur at the same position $\underline{R}^N$ and this gives rise to the following expression for the constant α

$$\alpha = \frac{(E_{bias} - V_0)(E_{bias} - V_{min})}{(V_{min} - V_0)} = \frac{3 N k_B T_{bias}(T_{bias} - T_{min})}{2 T_{min}}. \tag{5.10}$$

Under this scheme the user has easy control of the depth of the biased potential $V_{bias}(\underline{R}^N)$, to any depth between E_{bias} and V_0. Note that it does not matter if the system moves to a different potential basin (and hence adopts a different value of V_0), T_{bias} and T_{min} will be at the same heights above the minimum of the new basin. In this sense the the scheme outlined is adaptive with respect to the true energy surface.

As with the original scheme of Hamelberg *et al* [64] this prescription gives a system bias potential (5.4) which is everywhere differentiable (with continuous forces) and usefully retains some semblance of the topology of the original potential energy surface. The user is at liberty to chose any value $E_{bias} > V_0$, which is useful for configurational sampling, but for hyperdynamics satisfying the Voter condition (5.6) E_{bias} must not exceed the system configuration energy at any saddle point representing an escape route from a potential basin.

Finally it should be noted that simulations performed under the influence of a bias potential naturally do not return system averages corresponding to the thermodynamic state of the original system at the specified temperature and pressure. The calculation of the true thermodynamic

averages requires a correction in the form of a *weighted average* [64]. The true thermodynamic average $< A >$ of a property A is thus given by

$$< A >= \frac{< Ae^{\beta W_{bias}(\underline{R}^N)} >_{bias}}{< e^{\beta W_{bias}(\underline{R}^N)} >_{bias}} \tag{5.11}$$

Where the ensemble averages are obtained in the biased system. In practice, if the system visits many basins in the course of the simulation (i.e. passes through many states), it is not always sensible to compute such averages over the run. DL_POLY Classic only does this when using BPD to explore configuration space (see below, section 5.3.5), in which case V_0 is defined to be the *first* minimum visted in the simulation.

5.3.2 Running a BPD Simulation

Two ways of running bias potential dynamics are available in DL_POLY Classic. The first is referred to as "Full Path Kinetics" since it attempts to reproduce a full description of the diffusion path with the associated activation energies. This is described in section 5.3.3. (A variation of Full Path Kinetics allows the user to bypass the Nudged Elatic Band calculations, which is useful in circumstances where the NEB has problems.) The second is "configurational sampling", which exploits BPD to explore the range of structural states available to a system, which need not necessarily be in the solid state. It may also be used to improve thermodynamic averaging of a system at a given temperature, where equilibration is problematical due to long time scales. This is described in section 5.3.5.

5.3.3 Full Path Kinetics

This option is intended to determine the true diffusional path that a solid state system follows at a given temperature, but at an accelerated rate. Each time the system transforms from one structure to another (i.e. from one state to another) the program records the states it encounters and (optionally) calculates the activation energy E^* associated with the transition and extrapolates to the time at which the transition would have occured in the unbiased system. This information may subsequently be used to determine the full kinetics of the system.

The method in outline is as follows.

1. The first operation of the program is to construct a reference state for the structure by energy minimisation. The simulation then proceeds with the biased potential option in much the same manner as a normal simulation, but during which a running estimate of the boost factor (in equation (5.5)) is computed.

2. At user defined intervals (called here a 'BPD block') the simulation is halted and the structure energy minimised to create new reference structure, which is compared with the original reference state to determine if a transition has occurred. A transition is deemed to have occured if one or more atoms are displaced by more than a preset distance (the 'catch radius'). If a transition is detected, the program optionally performs a NEB calculation, using the two reference structures, to find the activation energy (E^*). Note that it is not essential calculate an activation energy if one is confident that the bias chosen does not exceed the safe limit described by Voter (equation 5.6).

3. A determination of the time of the transition is made. In DL_POLY Classic the occurrence time of the transition (t_{occ}) is determined by checking back from the detection of the transition through past configurations saved at regular intervals (which should be much less than a BPD

block). Each saved configuration is energy minimised and compared with the reference state structure until the first occurrence of the new state is found. This provides a reasonable accuracy on the transition time, somewhat better than using the end time of the BPD block in which the transition occurred. The transition time is then corrected for the boost factor in equation (5.5).

4. The new found state becomes the reference state for the next stage of the simulation. If no transition was detected the original reference state is left in place. In both cases the simulation continues from the end of the block as if uninterrupted. (Note this is markedly different from the TAD procedure described in section 5.4.)

5. The simulation is continued until, from inspection, it apparent that all significant kinds of transition have been observed. When this is is anybody's guess, but clearly some knowledge of the system, gained from other sources, it invaluable here.

6. With all the information gathered it should now be possible to determine the full diffusion process for the original system at the state point chosen.

The recommended procedure for running BPD with DL_POLY Classic is as follows.

1. Run a normal (unbiased) simulation of the system at the required state point (temperature and volume). Make sure the system does not undergo any structural changes that nullify the validity of the BPD approach (e.g. melting). Keep the REVCON file to use as the starting CONFIG structure for the BPD simulation.

2. Set up the BPD option in the CONTROL file as follows:

   (a) Set the **bpd path** directive.

   (b) Define the energy units for the BPD calculations e.g.
   units *s*
   where *s* is one of *eV, kcal, kJ or K*, signifying electron volts, kilo calories per mole, kilo joules per mole or Kelvin, respectively. No **units** directive means DL_POLY internal units apply. Forces are given in the chosen energy units per Angstrom.

   (c) Set the value of the potential bias (E_{bias}) e.g.
   ebias *f*
   where *f* is the bias energy level in Kelvin.

   (d) Set the value of the bias potential minimum (V_{min}) e.g.
   vmin *f*
   where *f* is the energy minimum in Kelvin.

   (e) If a BPD simulation *without* NEB calculations is required, set the 'no NEB' flag i.e.
   noneb

   (f) Select which atom type is to be tracked when determining transitions.
   target *atom_name*
   where *atom_name* is the name of the target atom type. The default, which is selected when this directive does not appear, is that all atom types are chosen.

   (g) Set the size of the simulation BPD block i.e. the number of time steps between structure optimisations (for transition detection). e.g.
   num_block 500.

(h) Set the number of configurations between each write of a tracking configuration file. This should be an integer divisor of the BPD block number. e.g.
num_track 10.

(i) Set the 'catch radius' i.e. the minimum distance in Angstroms any atom may be displaced in the minimised structure before it is recorded as a transition e.g.
catch_radius 3.0.

(j) Set the NEB spring constant (in specified energy units per $\mathring{A}^2$). e.g.
neb_spring 1000.0 (for DL_POLY units). This parameter is not required if the **noneb** flag has been set.

(k) Select a minimisation option. e.g.
force *key tol*.
Where *key* is one of *force, energy, position* and *tol* is the convergence tolerance. (The recommended choice is *force* with a tolerance of 1.0 in DL_POLY units.)

(l) Close the BPD definition with the directive
endbpd

3. Set other CONTROL file directives as follow:

   (a) Select the **restart noscale** option if the CONFIG file was pre-equilibrated, otherwise leave out the **restart** keyword altogether.

   (b) Set the length of the simulation required (**steps**) and the equilibration period (**equil**) (both in time steps). The equilibration can be short or absent if the system was pre-equilibrated. In which case a useful alternative is to choose one of the NVT algorithms.

   (c) In setting the job **close time**, it is recommended to set the number to at least 500 times the clock time it takes to do one normal MD time step. This is to prevent the program running out of time during a structural minimisation. The timing information for this may be taken from the previous equilibration run.

   (d) Set the remaining CONTROL keywords as were defined for the initial equilibration simulations.

4. Before starting the BPD simulation, use the UNIX 'mkdir' command to make the following empty directories:

   - *BASINS* - to receive any new structures found;
   - *TRACKS* - to store the tracking configurations;
   - *PROFILES* - to store any transition pathways found by NEB calculations.

   If the directories *BASINS, TRACKS* and *PROFILES* already exist then carefully archive the data before deleting the contents. These directories should not be emptied if the simulation is continuing (restarting) and a full history of the kinetics is required. More about these directories and the files they contain can be found in section 5.5

5. Run the BPD simulation. This will perform a simulation at the state point requested, checking for structural transitions at the BPD block intervals specified. Each time it finds a structural transition, it will record the new state, determine the activation energy by the NEB method (if requested) and the (unbiased) transition time using the boost factor in equation (5.5), and then continue the simulation.

6. When the simulation ends, proceed as follows.

(a) Check the EVENTS file to see if any structural transitions have been obtained. Each event is represented by a single record and transitions are flagged with the keyword TRA at the start of the record. Use unix 'grep' to locate these entries. No observed transitions indicates that either a longer simulation is necessary, or running with a higher bias E_{bias} should be considered.

(b) **Important.** If any of the reported transitions has a system activation energy that is below E_{bias} (i.e. $N * E^* < E_{bias}$, where N is the number of atoms in the system[1]) this represents a violation of the condition in equation (5.6), which means the observed diffusion path is not a valid representation of the original system. The simulation should be repeated with a lower value of E_{bias}.

(c) If the required number of time steps has not been reached, the simulation can be restarted from the REVCON, REVIVE and HYPRES files (renaming them as CONFIG, REVOLD and HYPOLD for the purpose), and setting the directive **restart** (with no qualifier) in the CONTROL file.

(d) Use the DL_POLY Java GUI to plot the system energy and temperature for the whole of the simulation. Apart from the equilibration period, these should hold their values within normal thermodynamic fluctuation, even if transitions have occured. If they do not, the system has probably not been equilibrated adequately to begin with, in which case the simulation should be started again.

(e) Check that all the new states the program found are present in the *BASINS* directory. Examine them using the DL_POLY Java GUI. There may be signs of imperfect minimisation (atoms not quite on lattice sites etc) but this is not a problem in this instance. More accurate NEB calculations can be performed later (see section 5.6) .

(f) Check that the profiles for all the reported transitions have been written in the *PROFILES* directory. These record the change in configuration energy as a function of reaction coordinate (or diffusion path). Do not do this if the option **noneb** was taken! Plot these using the DL_POLY Java GUI. Use the GUI 'spline' option to get a better idea of what the profiles look like. Take special note of any double (or multiple) maxima. The transition is considered to end at the first minimum in these cases. It follows that the activation energy for the second peak is not available in this case, but it can be obtained later by running the NEB facility independently for the states concerned (see section 5.6).

(g) It is useful to determine which atoms have relocated during a transition. The program bsncmp.f in the *utility* directory may be used for this purpose. It is designed to compare start and end configurations in the *BASINS* subdirectory and list the atoms that have changed location.

5.3.4 Things to Be Aware of when Running Full Path Kinetics BPD

1. Choose the 'catch radius' carefully, where possible basing it on nearest neighbour distances obtained from the parent crystal. A consequence of using too large a catch radius is that transitions that require a short hop in atom positions may be missed during a run. Such misses make it difficult to reconstruct the reaction path and, in particular, cause any NEB calculation to crash, since there is no simple path between the reference structures.

2. Note that in a BPD simulation the reference state is replaced whenever a new state is found. In this respect the reference state 'follows' the diffusion path. This is a clear distinction from TAD.

[1] Assuming just one atom undergoes the transition!

3. We repeat again the important message that if any of the transitions reported by PBD has an activation energy that is below the value of the bias term E_{bias} (i.e. $N * E^* < E_{bias}$,) this represents a violation of the condition in equation (5.6), which means the observed diffusion path is not a valid representation of the original system. The simulation should be repeated with a lower value of E_{bias}.

5.3.5 Exploring Configurational Space

Running DL_POLY Classic under the BPD option is useful for simply exploring configurational space. This has a number of uses:

1. Equilibration at a given temperature is quicker and thermodynamic averages can be obtained with greater reliability;

2. It is possible to observe configurations which are difficult to obtain under normal conditions, perhaps because they are far from the starting state and the system has slow relaxation times. Such configurations may be important from a mechanistic viewpoint;

3. The trajectory of the system evolves faster, which means that movies of the simulation can show the motions of the system on a reasonable time scale.

This option is activated in the CONTROL file by using the single-line directive:

bpd dyn f_1 f_2

where f_1 is the value of the required bias (E_{bias}), f_2 is the required value of the operating potential minimum ($< V_{min} >$) both expressed in Kelvin.

This option runs like a normal DL_POLY Classic simulation, except that the system potential is now the biased potential. Consequently average system properties are calculated using equation (5.11). (The user should note that the zero point of potential energy (V_0) in this case corresponds to the first energy minimum found in the simulation. It does not change when further minima are found. This is different from full path dynamics - see above.)

It is recommended that the simulation be run with the **traject** option activated in the CONTROL file so that a HISTORY file is produced. This may be further analysed to reveal conformational properties or viewed as a movie (with appropriate software).

5.4 Temperature Accelerated Dynamics

5.4.1 Theory of Temperature Accelerated Dynamics

Temperature Accelerated Dynamics (TAD) was devised by Voter *et al* [63]. Like BPD it is also a combination of molecular dynamics and Transition State Theory (TST) for first order processes. TAD works on the principle that while diffusion in the solid state at a low temperature is often too slow to measure, at a higher temperature it may be many orders of magnitude faster. However it is normally the case that at different temperatures a system will evolve via different diffusion pathways. So to exploit the temperature acceleration successfully special care must be taken to preserve the true mechanistic pathway at the required (low) temperature. This is precisely what TAD does.

An appropriate model for a first order diffusion process supposes a system trapped in a potential basin (state A), from which it may escape through thermal excitation to a new state (state B). If the system is created in state A at time zero, the probablity of it being found in the same state at a later time t is

$$P(t)dt = k \exp(-kt)dt \tag{5.12}$$

where $P(t)$ is a probability distribution and k is the first order rate constant. It follows from this that the mean lifetime (τ) of the system in state A is

$$\tau = 1/k \tag{5.13}$$

from which we have a universal property of first order systems

$$\tau\ k = 1. \tag{5.14}$$

In other words the rate constant is inversely proportional to the lifetime in the initial state.

According to TST the rate constant exhibits a temperature dependence given by the Arrhenius' law

$$k = \nu\ e^{\beta E^*} \tag{5.15}$$

where ν is the so-called the pre-exponential factor (with the units of frequency) and β is the Boltzmann factor $1/k_BT$. E^* is the activation energy of the process, which is the energy barrier between the bottom of the potential basin of state A and the saddle point on the energy surface that provides the escape route to state B. This equation shows that at different temperatures, T_1 and T_2, the same escape route from state A has different rate constants, k_1 and k_2 respectively. Nevertheless, the universal property of equation 5.14 means that

$$\tau_1\ k_1 = \tau_2\ k_2, \tag{5.16}$$

which is an important relation underpinning the TAD method, showing how the time scale for a barrier crossing event at one temperature is related to the time scale for the same event at another temperature.

In most practical systems state A is likely to have more than one escape route (to distinct states: B, C, D, etc.) each with its own activation energy, pre-exponential factor and temperature-dependent rate constant. At any given temperature, escape from state A may occur via any one of these routes, but is most probable via the route which has the highest rate constant and therefore (by equation 5.16) the lowest associated residence time. A normal molecular dynamics simulation commencing from state A will undergo a transition to a neighbouring state via the first encountered route and never sample the alternatives. Since the different routes have different temperature dependent rates, it follows that at different temperatures, the system may evolve along completely different paths. The TAD method avoids this possibility at high temperature by returning the system to state A after every transition, so that practically all of the escape routes at this temperature may be discovered. From the calculated properties of these escape routes the true low temperature escape route may be determined by extrapolation. Thus TAD provides a high temperature method for identifying the transitions that mark out the low temperature diffusion pathway.

The characteristics of the method are as follows, in which it is assumed that the kinetic properties of a system at the temperature (T_{low}) are required.

1. The starting structure (state A) is energy minimised to provide a reference structure (hereafter called the reference state) against which later structures may be compared to determine any structural transitions.

2. The system is simulated at high temperature (T_{high}) and halted at regular intervals (called a 'TAD block') to energy minimise the structure to construct a reference state. This is compared with the existing reference state to determine if a structural transition (to state B) has occurred. A transition is deemed to have occured if one or more atoms are displaced by more than a preset distance (the 'catch radius'). If a transition is detected, a NEB calculation is initiated, using the two reference structures, to find the activation energy (E^*).

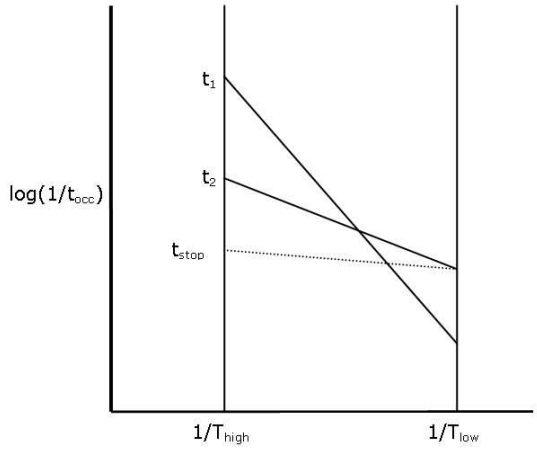

Figure 5.4: Basic TAD Theory
. Plot of $log(1/t)$ vs. $1/T$ for the TAD method. Simulations at high temperature locate transitions indicated as t_1 and t_2 with t_1 occurring first (time increases in a downward direction on this plot). Extrapolation to low temperature using equation 5.17 shows that these transitions would have occurred in reverse order. If no other transitions occurred, t_2 would be the observed low temperature transition in an MD simulation. The dotted line indicates a possible hypothetical transition that just precedes t_2 at low temperature. Its high temperature intercept is calculated according to the criterion of Voter *et al.* [63] which gives the estimated stopping time for the simulation.

3. Next a determination of the transition time (t_{occ}^{high}) is made. As with BPD the occurrence time of the transition (t_{occ}^{high}) is determined by checking back from the detection of the transition through past configurations saved at regular intervals (which are saved at intervals much less than a TAD block). Each saved configuration is energy minimised and compared with the reference state structure until the first occurrence of the new state is found. This provides a reasonable accuracy on the transition time, somewhat better than using the end time of the TAD block in which the transition occurred.

4. The time t_{occ}^{high} is extropolated to the corresponding time of occurrence (t_{occ}^{low}) at T_{low}. This is done by combining equations (5.15) and (5.16) and taking the logarithm:

$$\log\left\{\frac{t_{occ}^{low}}{t_{occ}^{high}}\right\} = \log\left\{\frac{k^{high}}{k^{low}}\right\} = -\frac{E^*}{k_B}\left\{\frac{1}{T_{high}} - \frac{1}{T_{low}}\right\}. \tag{5.17}$$

   See figure 5.4 for an indication of how the extrapolation works.

5. The system is returned to state A and the simulation recommenced. Returning the system to its original state means resetting the atomic coordinates to a structure in the starting basin and resetting the velocities according to a Boltzmann distribution, while retaining the total system energy of the original state. The simulation is continued to obtain information on other transitions (to states C, D, E etc) that may occur from state A. This is a key difference from the BPD method.

6. A determination of the simulation 'stopping time' (t_{stop}) is made (see below). When the simulation reaches the calculated stopping time, it is terminated.

When the simulation has ended, the transition with the shortest determined occurence time (t_{occ}^{low}) at T_{low} indicates the state to which the system would have transformed in a molecular dynamics simulation at that temperature. This new state becomes the starting point for a new high temperature simulation of the system, exploring transitions from this state to futher new states. By this procedure, after sufficient sampling of states, the true low temperature evolution of the system may be determined.

The 'stopping time' mentioned above is the time at which the high temperature simulation is halted. Ideally this is defined with a high probability that no more significant transitions will be found. This is determined from the history of the TAD simulation itself. Voter *et al.* provided a prescription of this [63]. It begins by defining, for a supposed undiscovered escape route, a very small probability (δ) that after the time t_{stop} the system is still in state A. This probability must chosen small enough to give confidence that the awaited transition has had sufficient time to occur. δ may be determined from

$$\delta = \int_{t_{stop}}^{\infty} k \ \exp(-kt) \ dt \tag{5.18}$$

from which it follows that

$$\log\left(\frac{1}{\delta}\right) = t_{stop}k. \tag{5.19}$$

and hence combining this with (5.15):

$$\log\left(\frac{1}{\delta}\right) = t_{stop}\nu_{min}\exp(-E_{min}^{*}/k_B T_{high}) \tag{5.20}$$

where ν_{min} and E_{min}^{*} are the prefactor and activation energy respectively of the supposed undiscovered escape route. Rearranging this gives

$$T_{high} \ \log\left(\frac{\log(1/\delta)}{t_{stop}\nu_{min}}\right) = -\frac{E_{min}^{*}}{k_B} \tag{5.21}$$

The supposed undiscovered escape route is one which may possesses a low temperature occurrence time that is less than the current working minimum (t_{occ}^{min}). The right side of (5.21) may be approximately determined using equation (5.17) if it assumed that the largest observed value of t_{occ}^{high} is close to t_{stop} and the lowest possible low temperature time is close to t_{occ}^{min} (see figure 5.4). Combining the two equations and rearranging gives

$$t_{stop} = \left(\frac{\log(1/\delta)}{\nu_{min}}\right)\left(\frac{t_{occ}^{min}\nu_{min}}{\log(1/\delta)}\right)^{T_{low}/T_{high}}. \tag{5.22}$$

Voter [63] argues that ν_{min} is commonly of the order $10^{12} \sim 10^{13}$ s^{-1} (or $1 \sim 10$ in DL_POLY units) and suggests $\delta = 0.001$ as a working value. These represent practical working values for approximating t_{stop}.

5.4.2 Running a TAD Simulation

This section describes the procedure for running a TAD simulation. The reader will notice some resemblance to the BPD procedure described in section 5.3. This is intentional for operational reasons, but the reader should always be alert to the key differences between the two.

We recommend the following procedure.

1. Run a normal simulation of the system at the (high) temperature (T_{high}) needed to perform the TAD simulation. Make sure the system behaves itself before moving to the next stage (and doesn't melt, for example). Retrieve the REVCON file and rename it as CONFIG for the TAD run. In principle this equilibration can be skipped and a TAD simulation started right away (with a suitable equilibration period at the start), but it is probably wiser to do this stage beforehand and make sure the system behaves properly at this temperature.

2. Set up the TAD simulation using the directives in the CONTROL file as follows:

   (a) Set the **tad** directive followed by records defining the operating conditions:

   (b) Define the energy units for the TAD parameters e.g.
   units *s*
   where *s* is one of *eV, kcal, kJ or K*, signifying electron volts, kilo cals per mole, kilo joules per mole or Kelvin, respectively. No **units** directive means DL_POLY internal units apply. Forces are in chosen energy units per Angstrom.

   (c) Set the size of the simulation TAD block i.e. the number of time steps between structure optimisations. e.g.
   num_block 500.

   (d) Set the number of configurations between each write of a tracking configuration file. This should be an integer divisor of the TAD block number. e.g.
   num_track 10.

   (e) Set the blackout period (in time steps) following a transition detection. e.g.
   blackout 200.
   A blackout period is intended to stop the program recording transitions that are correlated with a previous one. These are classified as 'ignored transitions'

   (f) Set the 'catch radius' i.e. the minimum distance in Angstroms any atom may be displaced in the minimised structure before it is recorded as a transition e.g.
   catch_radius 3.0.

   (g) Set the NEB spring constant (in specified energy units per $\mathring{A}^2$). e.g.
   neb_spring 1000.0 (in DL_POLY units).

   (h) Set the reliability factor for the high temperature simulation. For input purposes this is defined as the ratio $\log(1/\delta)/\nu_{min}$ (see above) e.g.
   deltad 0.001.

   (i) Set the low temperature (T_{low}) for the TAD method (i.e. the temperature for which the results are needed, in Kelvin) e.g.
   low_temp 30.0.

   (j) Select a minimisation option. e.g.
   keyword *tol*.
   Where **keyword** is one of **force, energy, position** and *tol* is the convergence tolerance. (The recommended tolerance for **force** option is 1.0 in DL_POLY units.) e.g.
   force 1.0.

   (k) Close the TAD definition with the directive
   endtad

3. Set other CONTROL file keywords as follow:

   (a) The simulation temperature (i.e. the 'high' temperature T_{high} for the TAD method) using the **temp** directive.

(b) Select the **restart noscale** option if the CONFIG file was pre-equilibrated, otherwise leave out the **restart** keyword altogether.

(c) Set the length of the simulation required (**steps**) and the equilibration period (**equil**) (both in time steps). The equilibration can be short if the system was pre-equilibrated.

(d) In setting the job **close time**, it is recommended to set the number to at least 500 times the clock time it takes to do one normal MD time step. This is to prevent the program running out of time during a structural minimisation. The timing information for this may be taken from the previous equilibration run.

(e) Set the remaining CONTROL keywords as were defined for the initial equilibration simulations.

4. Before starting the TAD simulation, use the UNIX 'mkdir' command to make the following empty directories:

   - *BASINS* - to receive any new structures found
   - *TRACKS* - to store the tracking configurations
   - *PROFILES* - to store any transition pathways found by NEB calculation

   If the directories *BASINS, TRACKS* and *PROFILES* already exist then carefully archive the data before deleting the contents. Do not empty these directories if continuing (restarting) the simulation in the original starting basin. The information in these directories is still 'live' in this case. Further information on these files can be found in section 5.5.

5. Run the TAD simulation. This will perform a simulation at the (high) temperature requested, checking for structural transitions at the intervals specified. Each time it finds a structural transition, it will record the new state, determine the activation energy, transition pathway and stopping time, then revert back to the starting basin and continue.

6. When the simulation ends, proceed as follows.

   (a) Check the EVENTS file to see if any structural transitions have been obtained. Each event is represented by a single record and transitions are flagged with the keyword TRA at the start of the record. Use unix 'grep' to locate these entries. No observed transitions indicates either a longer simulation is necessary, or a higher temperature simulation should be considered.

   (b) Check that the simulation was sufficiently long to guarantee all high temperature transitions have been found that are compliant with the specified reliability (**deltad**). The estimated stop time derived from this factor appears as the last entry of the TRA record in the EVENTS file.

   (c) If the simulation stop time has not been reached, the job must be restarted from the REVCON, REVIVE and HYPRES files (renaming them as CONFIG, REVOLD and HYPOLD for the purpose), and continued until the stop time has been reached. After the simulation finally stops, a new simulation can be started from the basin file obtained from the earliest (shortest extrapolated time) low temperature transition. See section 5.4.3 for more information on restarting a TAD simulation.

   (d) Use the DL_POLY Java GUI to plot the system energy and temperature for the whole of the simulation. Apart from the equilibration period, these should hold their values within normal thermodynamic fluctuation, even if transitions have occured. If they do not, the system probably has not been equilibrated adequately to begin with, in which case start the simulation again. (See sction 5.4.3.)

(e) Check that all the new states the program found are in the *BASINS* directory. Note that there may be fewer new states than the number of transitions observed because some transitions may end in the same basin more than once, so a new state is not stored in this case. Examine them using the DL_POLY Java GUI. There may be signs of imperfect minimisation (atoms not quite on lattice sites etc) but this is normal at this stage. Corrective action can be taken later (see section 5.6).

(f) Check that the profiles for all the reported transitions have been written in the *PROFILES* directory. These record the change in configuration energy as a function of reaction coordinate. Plot these using the DL_POLY Java GUI. Use the GUI 'spline' option to get a better idea of what the profiles look like. Take special note of any double (or multiple) maxima. The transition is considered to end at the first minimum in these cases. A basin file for the first intermediate state is written to the BASINS directory.

5.4.3 Restarting a TAD Simulation

It may be necessary to restart a TAD simulation for a number of reasons.

(a) The earliest low temperature transition from the current basin has been found and the user now wants to investigate transitions from the new basin. This basin corresponds to that which a molecular dynamics simulation would have reached first at the low temperature. In which case users should save their data from the first study and commence the simulation from the new basin exactly as in the previous study i.e. renaming the appropriate basin CFGBSNnn file as CONFIG. Once again an initial equilibration of the system to high temperature is recommended. This is not strictly a restart of an unfinished simulation, but the start of a new one which is part of a TAD series.

(b) The previous simulation ended but did not record any transitions. In this case it is advised to start the simulation afresh, using a higher operating temperature than before.

(c) The previous simulation ended without crashing and recorded some transitions but did not reach the required stop time. In this case simply restart the program as for a normal DL_POLY continuation run - using the REVCON, REVIVE and HYPRES files (renamed CONFIG, REVOLD and HYPOLD) and using the unqualified **restart** directive in the CONTROL file. (Remember to increase the number of required time steps if necessary.)

(d) The previous simulation crashed. If this means a crash for unknown reasons, then the situation may be unrecoverable (as with any unexpected DL_POLY crash). Try to locate the problem and fix it. If however the simulation arrived at this end point due to a time-out error, then there is hope. It may be possible to restart from the last REVCON, REVIVE HYPRES files, presuming they are uncorrupted and have the same time stamp. Be aware that such a restart may cause data duplication in other files, such as STATIS, EVENTS, *BASINS*, *PROFILES* and HISTORY, and the user should remove such a possibility by editing or sometimes even removing the files before restart. The objective is to remove any entries in these files that occured after the restart files were written. It is therefore important to determine what was going on when the program crashed. With TAD it may be found that the time-out error is most likely to happen during a NEB calculation or a structure optimisation. In which case it will be hard work deciding what needs to be patched up before continuing, though the time stamp of the restart files is still the crucial factor. This situation is best avoided in the first place by giving the code a generous 'close time' in the CONTROL file, so that these optimisation tasks have a chance to complete before the axe falls.

5.4.4 Things to Be Aware of when Running TAD

1. Choose the 'catch radius' carefully, where possible basing it on nearest neighbour distances obtained form the parent crystal. A consequence of using too large a catch radius is that transitions that require a short hop in atom positions may be missed during a run. Such misses make it difficult to reconstruct the reaction path and, in particular, cause the NEB calculation to crash, since there is no simple path between the reference structures.

2. The user may sometimes observe successive transitions into the same state. If a transition to an already visited state occurs it is indicated with the flag TRR (repeat transition) in the EVENTS file. Such repeated transitions are normal but if they occur in succession it implies that there is some correlation creeping into the resetting of the system back into the starting state. This however is harmless as the accumulated simulation time is reset back to the restart state after each transition and so does not affect the time of the later transition to a new state.

3. Note that in a TAD simulation the reference state is always the same. The reference state does not 'follow' the diffusion path as it does in BPD.

4. It is useful to determine which atoms have relocated during a transition. The program bsncmp.f in the *utility* directory may be used for this purpose. It is designed to compare start and end configurations in the *BASINS* subdirectory and list the atoms that have changed location.

5.5 DL_POLY Classic Hyperdynamics Files

The DL_POLY Classic BPD and TAD options generate a (potentially large) number of files in addition to those normally produced (and described in Chapter 4). Some are sufficient in number to warrant creation of additional sub-directories of the DL_POLY *execute* sub-directory. These files are as follows.

1. HYPRES - the hyperdynamics restart file, which stores (unformatted) data to permit continuation of an unfinished BPD or TAD simulation. It is created in the *execute* sub-directory. This file becomes the HYPOLD file, which is used in restarting a BPD or TAD simulation.

2. EVENTS - a summary of events that have occurred in the course of a hyperdynamics simulation - one record per event. It is generated in the *execute* sub-directory.

3. CFGBSNnn - a 'basin' file, which contains the coordinates of each distinct state DL_POLY Classic has found during the BPD or TAD run. nn is an integer rising from 0 to 9999. All such files are generated in the *execute/BASINS* sub-directory.

4. PROnn.XY - a 'profile' file, which is a list of the reaction coordinate and configuration energy of each bead in the converged NEB calculation. nn is an integer rising from 0 to 9999. All such files are generated in the *execute/PROFILES* sub-directory and are plotable XY files.

5. CFGTRAnn - a configuration file used to interpolate when a transition has occured. nn is an integer rising from 0 to 9999. All such files are generated in the *execute/TRACKS* sub-directory.

These files are described in further detail below.

5.5.0.1 The HYPRES and HYPOLD Files

The HYPRES and HYPOLD files are unformatted (i.e. not human readable) and are *restart files* for BPD or TAD runs of DL_POLY Classic. The HYPRES file is produced by the program at regular intervals during the program run and also at the end of a run. It must subequently be renamed HYPOLD to be read by DL_POLY Classic when the simulation is recommenced. The user does not need to know the contents of these files, but for the curious it can be said that they contain current file numbers for the *BASINS*, *TRACKS* and *PROFILES* directories; the structural differences between the current reference basin and any new basins found (TAD only); and the atomic coordinates of the current basin taken at the last check point (such as the end of the last BPD orTAD block).

5.5.0.2 The EVENTS File

The EVENTS file is a text file that reports the results of actions taken by the hyperdynamics routines. Each record in the file specifies a particular kind of event. The possible events described are as follows. (Note that the real variables specified in this file are in units specified by the user.)

1. Blackout period reset: **BLK n1 n2**
   where
   - n1 is the time step at which a blackout period was initiated;
   - n2 is time step at which the new blackout period will end.

   TAD only.

2. Equilibration period reset: **EQL n1 n2**
   where
   - n1 is the time step at which the equilibration period was reset;
   - n2 is time step at which the new equilibration period will end.

3. Minimisation completed: **MIN n1 n2 n3 n4 r1 r2 r3**
   where
   - n1 is the time step at which the minimisation commenced (integer);
   - n2 is number of cycles required by the minimiser to converge (integer);
   - n3 is the BPD/TAD block for which the minimisation took place (integer);
   - n4 is the optimisation convergence criterion key: 0 for forces, 1 for energy, 2 for position (integer);
   - r1 is the convergence tolerance used by the minimiser (real);
   - r2 is the energy of the minimised configuration (real);
   - r3 is the the convergence actually achieved by the minimiser (real).

   Users should note that a final convergence value (r3) greater than the convergence criterion (r1) indicates incomplete convergence.

4. Nudged Elastic Band completed: **NEB n1 n2 n3 n4 r1 r2**
   where
   - n1 is the time step at which the NEB calculation commenced (integer);

- n2 is number of cycles required by the NEB calculation to converge (integer);
- n3 is the maximum allowed number of cycles (integer);
- n4 is the number of 'beads' in the NEB chain (integer);
- r1 is the energy of the home basin (starting state) configuration (real);
- r2 is the energy of the end basin (new state) configuration (real);

Users should note that when n2 and n3 are equal, this implies that convergence of the NEB chain has not been achieved. Note also that the characteristics of the reaction path are given by the subsequent **TRA** event (below).

5. Transition detected: **TRA n1 n2 n3 n4 r1 r2 r3 r4**
where

- n1 is the time step at which the transition was first detected (integer);
- n2 is the home basin (starting state) of the transition (integer);
- n3 is the new basin (ending state) of the transition (integer);
- n4 is the number or turning points in the transition profile (integer);
- r1 is the activation energy obtained from the NEB calculation (real);
- r2 is the observed transition time (real);
- r3 is the calculated extrapolated transition time (real);
- r4 is the calculated stopping time (real, TAD only);

Note that iof the user has set the **noneb** directive in a BPD simulation the entry for the activation energy is replaced by the bias potential, which sets a lower bound for the activation energy.

6. Transition ignored: **TRI n1**
where

- n1 is the time step at which a transition was detected, but ignored because it was during an equilibration or blackout period (integer).

TAD Only.

7. Transition repeated: **TRR n1 n2 n3**
where

- n1 is the time step at which a transition was detected, but it was identified as a repeat and no further analysis was underatken (integer);
- n2 is the identity of the home basin (integer);
- n3 is the identity of the new basin (integer);

TAD Only.

5.5.0.3 The CFGBSNnn Files in the *BASINS* Directory

A CFGBSNn file is a text file containing the energy minimised structure of a basin found during the BPD or TAD simulation. The number nn rises from 0 to 9999. Internally the format of the file is the same as a CONFIG file (see section 4.1.2), though it does not normally contain velocity or force data.

5.5.0.4 The CFGTRKnn Files in the *TRACKS* Directory

The CFGTRKnn files have exactly the same format as the CFGBSNnn files. The files do not however contain energy minimised structures. These files represent consecutive structures written at user defined intervals during the simulation. The interval (**num_track** see above) is an integer divisor of the number of steps in a BPD or TAD block (**num_block**) and the number nn in the file name is modulo **max_track**, where **max_track=num_block/num_track**. Thus after **num_block** time steps from the simulation start, there are always **max_track** configurations to search back over to locate the time of a transition. nn is an integer ranging from 0 to **max_track**.

5.5.0.5 The PROnn.XY Files in the *PROFILES* Directory

The PROnn.XY files tabulate the converged configuration energies of the beads in a NEB calculation, as a function of the reaction coordinate linking the beads. nn is an integer ranging from 0 to 9999.

The reaction coordinate is the path distance (S_n) between the structure of the reference state and the structure of a converged NEB bead and is defined here as:

$$S_n = \sum_{i=1}^{n} \left[(\underline{R}_i^N - \underline{R}_{i-1}^N)^2 \right]^{1/2}, \tag{5.23}$$

where $\underline{R}_i^N$ is a 3N dimensional vector defining the structure (N is the number of atoms) and n ranges from 2 to bead number N_{neb} in the NEB chain. Note that the reaction path does not usually represent a straight line in the 3N dimensional space. The file PROnn.XY presents two columns of numbers: the first is the reaction coordinate and the second is the configuration energy of the bead. Both are expressed in DL_POLY Classic units. The configuration energy for the first bead (at $S_n = 0$) is the energy of the reference state.

Normally the PROnn.XY file reveals a single maximum in configuration energy as the reaction coordinate increases. However in some instances more than one maximum may be obtained. The user should note that in these instances DL_POLY Classic will take the configuration closest the first minimum and optimise it independently to define the true destination of the transition from the reference state.

5.6 Tidying Up the Results of a Hyperdynamics Simulation

5.6.1 Refining the Results

A completed BPD or TAD simulation will provide a number of basin files defining the minima of new structures discovered, together with the associated profile files describing the energy path between these structures. These are the data that are needed to reconstruct the diffusion path in the original system.

However, at this stage there are still some approximations in the results, which arise from the chosen tolerances in the energy minimisation of the structures and the NEB calculations. To offset these, the following refinements are recommended.

1. Take each of the basin structures derived from the BPD or TAD simulation and perform a further structural optimisation with DL_POLY, using more exacting convergence tolerance. For example using a force tolerance of 0.01 (DL_POLY units) in place of the recommended 1.0 used in the BPD and TAD procedures. This will provide more accurate reference structures.

2. Using the accurately minimised structures in place of the original basins, use the NEB option in DL_POLY to recalculate the transition path between the reference states. Once again a more exacting tolerance may be used, but beware that the NEB calculation may not converge at all if the tolerance is too exacting. It is far less stable in this respect than the ordinary structural optimisation. Note that the tolerance for the overall NEB minimisation is set internally in DL_POLY Classic to be a factor of 10 larger than that for the minimisation alone. The result of these refinements should be a better estimate of the activation energy and low temperature transition time.

For TAD simulations the activation energy obtained from the refined structures can be used together with the simulated high temperature transition time to recalculate the low temperature transition time from equation (5.17). Note this may alter the original low temperature diffusion path, so be alert to this possibility and change the starting basin for any subsequent simulation. These refinements have no impact on the BPD simulations other than to improve the quality of the calculated kinetic properties.

5.6.2 Treatment of Multiple Maxima in the Reaction Path

The NEB calculations that occur while the BPD or TAD simulations are running may sometimes report a multiple maximum on the reaction path (see the **TRA** entry for the EVENTS file in section 5.5.0.2). More than two maxima is probably indicative of problems with the NEB convergence and should be regarded with suspicion, but obtaining two maxima is a real possibility. In such cases DL_POLY Classic stores both the end structure of the NEB chain and the structure corresponding to the first minimum in the energy profile along the reaction path, but it does not record the activation energies beyond the first peak.

A BPD simulation requires a complete description of the potential energy surface kinetics so determination of the second activation energy and the corresponding transition time is essential. After recording the transition, the subsequent dynamics correctly starts (for BPD) from the final state of the double transition, but the loss of information is ignored. A NEB calculation is therefore necessary to determine the lost details.

For TAD objective is to find the escape route for a transition from the starting state and halting the analysis of the reaction path at the first minimum is sufficient to define the escape. The intermediate state provides a valid possible basin for further study of the kinetics of the system. This is sensible if the two peaks on the reaction path are of similar magnitude. However it is quite possible that the second peak is much higher or much lower than the first. The first of these possibilities means that choosing the first minimum as the starting basin for a new simulation will most likely consistently return the system to the original starting state. The second possibility suggests that the second state on the reaction path is a better option for the next phase of the study. To decide between these possibilities it is necessary to determine the activation energy of the second peak. Thus in both BPD and TAD, when a multiple maximum is found on the reaction path, a NEB calculation is needed to complete the path analysis.

See the following section 5.7 for details.

5.7 Running a Nudged Elastic Band Calculation

Running an independent NEB calculation may be necessary to improve the accuracy of the calculated activation energy, or to determine the activation energy in transitions not fully evaluated when they occurred in a BPD or TAD simulation - due to the occurrence of a multiple maximum on the reaction path.

To run a NEB calculation with DL_POLY Classic it is first necessary to identify the start and end basins among the CFGBSNnn files in the BASINS directory described in section 5.5.0.3. From the information provided in the EVENTS 5.5.0.2 file it should be possible to decide which files are needed. The user then needs to modify the CONTROL file in the following way.

1. Remove any directives for the **bpd** or **tad** options. Directives for the integration algorithm (**integrator**) or ensemble (**ensemble**) should also be removed.

2. The directive for the NEB option should be inserted:
   neb *n*
   where *n* is the number of NEB calculations required.

3. On the record following the **neb** directive, a list of *n* starting basins should be given e.g.
   basin_1 1 1 1 2 3
   Meaning the 5 required NEB calculations start from basin files CFGBSN0001, CFGBSN0001, CFGBSN0001, CFGBSN0002 and CFGBSN0003. Up to 10 NEB calculations are permitted.

4. On the second record following the **neb** directive, a list of *n* final basins should be given e.g.
   basin_2 2 3 4 3 4
   Meaning the 5 required NEB calculations are between basins 1 - 2, 1 - 3, 1 - 4, 2 - 3 and 3 - 4 in this example.

5. Define the energy units for the BPD parameters e.g.
   units *s*
   where *s* is one of *eV, kcal, kJ or K*, signifying electron volts, kilo cals per mole, kilo joules per mole or Kelvin, respectively. No **units** directive means DL_POLY internal units apply. Forces are in chosen energy units per Angstrom.

6. Next set the NEB spring constant (in specified energy units per $\mathring{A}^2$). e.g.
   neb_spring 1000.0 (in DL_POLY units).

7. Select a minimisation option. e.g.
   force *key tol.*
   Where *key* is one of *force, energy, position* and *tol* is the convergence tolerance.

8. Close the NEB definition with the directive
   endneb

5.7.1 Things to Aware of when Running a NEB Calculation

1. Note that the NEB calculation assumes that the basin files for the start and end states are in the BASINS directory and that DL_POLY Classic is being run from the *execute* directory, where the DLPOLY.X executable is located. Needless to say, if these files are placed anywhere else, the calculation will fail.

2. Note also that the NEB calculation places the reaction path profile for a given pair of states in the PROFILES direction with the file name PRXnn.XY, where nn is a *negative* number that is compounded from the identities of the start (n_1) and end states (n_2) thus: $nn = -(100 * n_1 + n_2)$.

3. It is important to be sure that the start and end states represent real, observed transitions in the BPD or TAD simulation. The danger here is using two structures that are not mechanistically close. If this is not the case, the NEB calculation is unlikely to converge, as there

will be no simple path (with preferably a single energy maximum) between the start and end states.

4. When running an NEB calculation to improve the accuracy of the activation energy a more stringent tolerance must be set. For example the recommended value for the force tolerance is normally 1.0 (in DL_POLY units), but values one or two orders of magnitude less may be tried. It should be noted however that before the NEB calculation is run, the configurations representing the start and end configurations must first be minimised to the accuracy required by the new tolerance, by using the DL_POLY Classic **optim** option. These optimised structures must be returned to the BASINS directory with the same file numbers as the original CFGBSN files.

Chapter 6

Solvation

Scope of Chapter

This chapter describes the features within DL_POLY Classic relevant to the subject of solvation. The main features are: decomposing the system configuration energy into its molecular components; free energy calculations by thermodynamic integration; and the calculation of solvent induced spectral shifts. Some of these features are sufficently general to have applications in other areas besides solutions.

6.1 Overview and Background

This chapter is about the features within DL_POLY Classic for studying solutions. These include decomposing the system configuration energy into various molecular components, calculating free energies by the method of thermodynamic integration, and calculating solvent induced spectral shifts. Despite the focus on solutions however, some of these features are applicable in other scientific areas. In particular the energy decomposition can be employed in any system of mixed species and the free energy feature can be used for other systems where a free energy difference is required.

A DL_POLY Classic module SOLVATION_MODULE.F has been devised for these purposes. It was developed in a collaboration between Daresbury Laboratory and the Institut Pluridisciplinaire de Recherche sur l'Environment et les Materiaux (IPREM), at the University of Pau. The collaborators included Ross Brown, Patrice Bordat and Pierre-Andre Cazade at Pau and Bill Smith at Daresbury. The bulk of the software development was done by Pierre-Andre Cazade and was extended and adaptated for general DL_POLY distribution by Bill Smith.

6.2 DL_POLY Energy Decomposition

6.2.1 Overview

In DL_POLY Classic the energy decomposition capability breaks down the system configuration energy into its contributions from the constituent molecular types. What consitutes a molecule in this context is what is defined as such in the DL_POLY Classic FIELD file (see section 4.1.3). It is not essential that all the atoms in the molecular definition be linked together by chemical bonds. Nor is it essential for all identical molecules to be declared as one molecular type. Groups of like molecules or individual molecules can be separated out if there is a compelling reason to do so. It is not however possible to split molecules that are linked by chemical bonds into sub-molecules for this purpose.

The configuration energy decomposition available in DL_POLY Classic can be summarised as follows.

1. For each **unique** molecular type in the system (A, B, C *etc*) the program will calculate:
   - the net bond energy;
   - the net valence angle energy;
   - the net dihedral angle energy;
   - the net inversion angle energy; and
   - the net atomic polarisation energy.

   These are the so-called intramolecular interactions, while those below are considered to be intermolecular.
2. For each **unique pair** of molecular types in the system (AA, AB, AC, BB, BC, CC *etc*) the program will calculate:
   - the net coulombic energy; and
   - the net Van der Waals energy.
3. For each **unique triplet** of molecular types in the system (AAA, AAB, AAC, ABB, ABC, ACC *etc*) the program will calculate:

- the net three-body angle energy.

4. For each **unique quartet** of molecular types in the system (*AAAA*, *AAAB*, *AACC*, *AABB*, *AABC*, *AACC etc*) the program will calculate:

   - the net four-body angle energy.

These are calculated at user-specified intervals in the simulation from a chosen starting point and written to a data file called SOLVAT, which is decribed in section 6.2.3 below. It is not required that all the above different kinds of interaction are present in the same system. The types of intramolecular interaction that may be defined in DL_POLY Classic are described in section (2.2). The types of intermolecular terms that can be defined are described in section (2.3) and the Coulombic methods available are described in section (2.4).

Note that all the interaction types that are classed as intermolecular above may occur as intramolecular interactions if the molecule concerned is defined as including them. Nevertheless they are counted as intermolecular terms for the purposes of summation by DL_POLY Classic. It should also be noted that for technical reasons, the program cannot supply the Coulombic decomposition if the SPME option (section 2.4.7) is selected, but the standard Ewald option is valid for this purpose. Furthermore, there is no decomposition available for metallic potentials (section 2.3.5) or the Tersoff potential (section 2.3.3), since these are many-body interactions not readily amenable to simple decomposition.

6.2.2 Invoking the DL_POLY Energy Decomposition Option

The energy decomposition option is activated when the appropriate directive is inserted into the CONTROL file (section 4.1.1). The directive may be either **decompose** or **solvate**, which have the same effect, though the user's purpose in invoking each is different. Acceptable abbreviations of these directives are **decomp** or **solva**.

The simplest form of invocation is a single line entry:

decompose `n1 n2`
or
solvate `n1 n2`

where the number `n1` specifies the time step at which DL_POLY Classic is to start calculating the required data, and `n2` is the interval (in time steps) between calculations of the data.

The invocation may also be made in a more informative way:

decompose (or **solvate**)
start `n1`
interval `n2`
enddec (or **endsol**)

in which **start** and **interval** specify the start time step (`n1`) and time step interval (`n2`) respectively. Directives **enddec** or **endsol** close the data specification.

6.2.3 The SOLVAT File

The SOLVAT file is a file in which DL_POLY Classic writes all the energy decomposition/solvation data. It is an appendable file and is written to at intervals (defined by the user) during the

simulation. Restarts of DL_POLY Classic will continue to append data to an existing SOLVAT file, so the user must be careful to ensure that this is what is actually wanted.

Its contents are as follows.

record 1 Format (80a1): The job title, as defined at the top of the CONTROL file.

record 2 Format (40a1): Energy units as defined in the FIELD file header.

record 3 Format (2i10): (**natms**, **mxtmls**) - the numbers of atoms and molecule types in system.

record 4 Format (1x,11a4): Information record - labels contents of record 5:
lex lsw bnd ang dih inv shl cou vdw 3bd 4bd.

record 5 Format (11L4): **lexcite**, **lswitch**, **lcomp(1-9)**. Logical control variables, where each indicates the following:
lexcite = `.true.` - spectroscopic (excited state) calculation (see section 6.4.2);
lswitch = `.true.` - switching between states for solvent relaxation study (see section 6.4.3);
lcomp(1) = `.true.` - bond energies are present;
lcomp(2) = `.true.` - valence angle energies are present;
lcomp(3) = `.true.` - dihedral angle energies are present;
lcomp(4) = `.true.` - inversion angle energies are present;
lcomp(5) = `.true.` - atomic polarisation energies are present;
lcomp(6) = `.true.` - coulombic energies are present;
lcomp(7) = `.true.` - van der Waals energies are present;
lcomp(8) = `.true.` - 3-body energies are present;
lcomp(9) = `.true.` - 4-body energies are present;

record 6 - end-of-file Format (5e14.6): All subsequent records list the calculated data in individual blocks for each requested time step. Each block record may consist of any, or all, of the following data records, depending on the system being simulated (as indicated by **record 5**). Note that individual data records may require more than one line of the SOLVAT file since only five real numbers are presented on each line. In simulations where solvation induced shifts studies are being performed (i.e. where the control variable **lexcite** is set `.true.` - see section (6.4.5)), each of the data records is duplicated, thus providing data for the ground and excited state systems separately (see section 6.4.2). In the following `mxtmls` represents the number of molecule types in the system.

block record 0: species temperatures (mxtmls entries)

block record 1: bond energies (mxtmls entries)

block record 2: valence angle energies (mxtmls entries)

block record 3: dihedral angle energies (mxtmls entries)

block record 4: inversion angle energies (mxtmls entries)

block record 5: atomic polarisation energies (mxtmls entries)

block record 6: coulombic energies ((mxtmls(mxtmls+1))/2 entries)

block record 7: van der Waals energies ((mxtmls(mxtmls+1))/2 entries)

block record 8: 3-body energies ((mxtmls(2+mxtmls(3+mxtmls)))/6 entries)

block record 9: 4-body energies ((mxtmls(6+mxtmls(11+mxtmls(6+mxtmls))))/24 entries)

It should be noted that writing the 2-, 3- and 4-body energies as a linear stream implies a certain ordering of molecule pairs (indices i,j), triplets (indices i,j,k) and quartets (indices i,j,k,m). The appropriate sequence order can be reconstructed from simple nested loops for pair, triple or quadruple indices subject to the conditions: i≥j for pairs; i≥j≥k for triplets; and i≥j≥k≥m for quartets, with i as the outermost loop index. For example the following code generates the correct sequence for a quartet as variable `index`.

```
index=0
do i=1,mxtmls
  do j=1,i
    do k=1,j
      do m=1,k
         index=index+1
      enddo
    enddo
  enddo
enddo
```

To assist users with analysis of the SOLVAT file two utility programs are available in the *utility* directory. The program **solsta.f** will calculate the averages and RMS deviations for all the variables in the file and the program **soldis.f** will construct the distribution functions of all the variables in a form suitable for plotting.

6.3 Free Energy by Thermodynamic Integration

6.3.1 Thermodynamic Integration

Thermodynamic Integration (TI) is a well established method for calculating the free energy difference between two systems defined by distinct Hamiltonians H_1 and H_2. A 'mixed' Hamiltonian is defined with the aid of a mixing parameter λ, where $0 \leq \lambda \leq 1$, as follows:

$$H_\lambda = (1-\lambda)H_1 + \lambda H_2, \tag{6.1}$$

so that when $\lambda = 0$ the Hamiltonian corresponds to system 1 and when $\lambda = 1$ it corresponds to system 2. Intermediate values λ mix the two systems in different proportions. From such a Hamiltonian and the relationship between the free energy F and the system partition function it is easy to show that

$$\frac{dF}{d\lambda} = \langle H_2 - H_1 \rangle_\lambda \,. \tag{6.2}$$

From this it follows that if the average of the difference $(H_2 - H_1)$ is calculated from a series of simulations over a range of λ values (between 0 and 1), it is possible to integrate this equation numerically and obtain the free energy difference between systems 1 and 2 i.e.

$$\Delta F_{12} = \int_0^1 \langle H_2 - H_1 \rangle_\lambda \, d\lambda. \tag{6.3}$$

Though simple in principle, there are two problems with the basic technique.

1. Firstly, if the mixed Hamiltonian requires the kinetic energy components to be scaled (by either λ or $1-\lambda$) the the equations of motion become unstable when λ approaches either 0 or 1. (This is because scaling the kinetic energy components amounts to a rescaling of the atomic masses, which can thus approach zero at the extremes of λ. Near zero mass dynamics is not stable for normal time steps.) Fortunately, it is possible in many cases to set up the Hamiltonians H_1 and H_2 so that the mixed Hamiltonian does not require scaling of the kinetic energy. In these cases there is no problem with the equations of motion. For the awkward cases where the kinetic energy really must be scaled, DL_POLY Classic has the option **reset_mass**, which scales the masses as required. Ideally however, this circumstance should be avoided if at all possible.

2. Secondly, it is well known that when λ approaches 0 or 1, the average $\langle H_2 - H_1 \rangle_\lambda$ in equation (6.3) is subject to large statistical error. This arises because the modified dynamics of the mixed Hamiltonian permits unnaturally close approaches between atoms, and the configuration energy terms arising from this are inevitably extremely large. Fortunately this problem can be mitigated by the use of a suitable weighting function, examples of which are described in the following section.

As an example of how this approach maybe used, we present the calculation of the free energy of a solution of a solute A in a solvent S. An appropriate choice of Hamiltonians for this is

$$\begin{aligned} H_1 &= K_S + K_A + V_{SS} + V_{AA} + V_{AS} \\ H_2 &= K_S + K_A + V_{SS} + V_{AA} \end{aligned} \tag{6.4}$$

in which K_S and K_A are the kinetic energies of the solvent and solute respectively, V_{SS}, V_{AA} and V_{AS} are the interaction energies between solvent-solvent, solute-solute and solute-solvent molecules respectively. Hamiltonian H_1 contains a term (V_{AS}) that causes the solvent and solute to interact, while H_2 has no such interaction. The mixed Hamiltonian in this case is

$$H_\lambda = K_S + K_A + V_{SS} + V_{AA} + (1-\lambda)V_{AS}. \tag{6.5}$$

It will be appreciated that when λ is 0, this Hamiltonian represents the solution of A in S, and when λ is 1, it represents complete independence of the solute and solvent from each other. The Hamiltonian thus encapsulates the process of solvation. It should also be noted that there is no scaling of the kinetic energy in this case, so instabilities are not expected in the dynamics when λ is near 0 or 1.

Following the above prescription we see that in this case

$$\frac{dF}{d\lambda} = \langle V_{AS} \rangle_\lambda \,. \tag{6.6}$$

and

$$\Delta F_{12} = -\int_0^1 \langle V_{AS} \rangle_\lambda \, d\lambda. \tag{6.7}$$

This equation represents the free energy difference between the free solvent and solute and the solution. The quantity $-\Delta F_{12}$ is thus the free energy of solution.

6.3.2 Nonlinear Mixing

As mentioned above, the linear mixing of Hamiltonians represented by equations (6.1) and (6.5) gives rise to poor statistical convergence of the required averages when λ approaches either 0 or 1.

One way to reduce the effect this has on the quality of the free energy calculation is to introduce weighting into the averaging process, so that poor convergence of the averages at the extremes of λ is of less importance.

This is done by defining a more general form for the mixing as follows

$$H_\lambda = (1 - f(\lambda))H_1 + f(\lambda)H_2, \tag{6.8}$$

in which $f(\lambda)$ is an appropriately designed function of the mixing parameter λ. In this case the derivative of the free energy with respect to λ is

$$\frac{dF}{d\lambda} = \left\langle (H_2 - H_1)\frac{df(\lambda)}{d\lambda} \right\rangle_\lambda . \tag{6.9}$$

As before this equation may be integrated to give the free energy difference as in equation (6.3).

It should now be apparent what desirable properties $f(\lambda)$ needs to have. Firstly it should be zero when $\lambda = 0$ and unity when $\lambda = 1$. Secondly the derivative of the function should approach zero when λ approaches either 0 or 1, where it will diminish the contribution of the extremes of the integral to the overall result. With these requirements in mind, DL_POLY Classic has a number of options for the function $f(\lambda)$.

1. Standard linear mixing:
$$f(\lambda) = \lambda, \tag{6.10}$$

2. Nonlinear mixing:
$$f(\lambda) = 1 - (1 - \lambda)^k, \tag{6.11}$$
where k is an integer exponent;

3. Trigonometric mixing:
$$f(\lambda) = \frac{1}{2}(1 + sin(\pi(\lambda - \frac{1}{2}))), \tag{6.12}$$

4. Error function mixing:
$$f(\lambda) = \frac{\alpha}{\sqrt{\pi}} \int_0^\lambda exp(-\alpha^2(x - \frac{1}{2})^2)dx, \tag{6.13}$$
where α is a parameter of order $10 \sim 11$;

5. Polynomial mixing:
$$f(\lambda) = 1 - (1 - \lambda)^k \sum_{i=0}^{k-1} \frac{(k - 1 + i)!}{(k - 1)!i!}\lambda^i, \tag{6.14}$$
where k is an integer exponent.

6. Spline kernel mixing:
$$f(\lambda) = 2\lambda - 8(\lambda - 1/2)^3(1 - abs[\lambda - 1/2]) - 1/2, \tag{6.15}$$

All these functions except (6.10 and 6.11) have the required properties (though not all are equally effective). Function (6.11) is suitable for mixed Hamiltonians which have only one problematic end point, such as (6.5), when $\lambda \sim 1$.

6.3.3 Invoking the DL_POLY Free Energy Option

The free energy option using thermodynamic integration is activated by the directive **free** in the CONTROL file (section 4.1.1). This is followed by additional directives on the following lines, terminating with the directive **endfre**. The invocation is therefore made in the following way:

free
start `n1`
interval `n2`
lambda `r1`
mix `n3`
expo `n4`
reset_mass - (this is not recommended)
system_a `i1 i2`
system_b `i3 i4`
endfre

The meaning of these directives is as follows.

1. **free** - invokes the free energy option and marks the start of the free energy specification in the CONTROL file;
2. **start** `n1` - specifies the time step at which DL_POLY Classic should start producing free energy data (integer `n1`) ;
3. **interval** `n2` - specifies the time step interval between free energy data calculations (integer `n2`);
4. **lambda** `r1` - value of the mixing parameter λ in the Hamiltonian (equation (6.1), range 0-1 (real `r1`);
5. **mix** `n3` - key for choice of mixing protocol (integer `n3`), choices are:
   - `n3=1`, linear mixing (equation (6.10));
   - `n3=2`, nonlinear mixing (equation (6.11));
   - `n3=3`, trigonometric mixing (equation (6.12));
   - `n3=4`, error function mixing (equation (6.13));
   - `n3=5`, polynomial mixing (equation (6.14));
   - `n3=6`, spline kernel mixing (equation (6.15));
6. **expo** `n4` - exponent for nonlinear or polynomial mixing, as in equations (6.11)and (6.14) respectively (required for these options only) (integer `n4`);
7. **reset_mass** - if this flag is present the Hamiltonian mixing will include the kinetic energy. The default (obtained by removing this flag) is that there is no mixing of the kinetic energy.
8. **system_a** `i1 i2` identifies the range of atom indices that consitute the species A in the CONFIG file (integer `i1`,`i2`);
9. **system_b** `i3 i4` identifies the range of atom indices that consitute the species B in the CONFIG file (integer `i3`,`i4`);
10. **endfre** - closes specification of free energy.

The invocation of the free energy option means that DL_POLY Classic will produce an output file named FREENG, the contents of which are described in section (6.3.4) below.

Some further comments are in order. Firstly it should be noted that the solute species A and B, which are specified using directives **system_a** and **system_b**, are identified by specifying the range of atom indices these components have in the CONFIG file (see section 4.1.2). It is apparent that this allows the user to specify atoms in the categories of A and B which are not related to underlying molecular structures. This is a simple way of identifying the distinct components of the combined system. However there is a clear need for the user to be cautious in defining the system if 'strange' simulations are to be avoided. Also it is apparent that atoms that do not fall under the categories of A or B will be deemed to be solvent atoms. It is not really sensible to specify all atoms as either A or B, with no solvent atoms (in category S) at all. In some circumstances A and B may have atoms in common, in which case these can be be treated as part of the solvent without affecting their physical properties. It is permissible to have no atoms in category A or category B if that is required.

6.3.4 The FREENG File

The FREENG file is a formatted in which DL_POLY Classic writes all the free energy data requested. It is an appendable file and program restarts will continue to add data to it if it already exists. The data are written at user-defined intervals during the simulation. The contents of the file are as follows.

record 1 Format (80a1): The job title, as defined at the top of the CONTROL file.

record 2 Format (40a1): Energy units as defined in the FIELD file header.

record 3 Format (4e16.8) **lambda, lambda1, lambda2, dlambda**, with

- **lambda** - Hamiltonian mixing parameter λ (real);
- **lambda1** - mixing factor $(1 - f(\lambda))$ (real);
- **lambda2** - mixing factor $f(\lambda)$ (real);
- **dlambda** - derivative of **lambda1** w.r.t. λ.

record 4 - end-of-file Format (i10,2e16.8) **nstep, engcfg, vircfg**, where

- **nstep** - time step of data (integer);
- **engcfg** - configuration energy difference $[V_2 - V_1]$ (real);
- **vircfg** - virial difference $[\Psi_2 - \Psi_1]$. (real).

The configuration energy presented in the FREENG file may be averaged over the entire run to obtain the configuration energy contribution to the average on the right of equation (6.1). The virial presented there may be used in the calculation of the Gibbs free energy, but it is not needed for Helmholtz (NVT) free energies. Note that the configuration energy and virial differences *include* the factor $df(\lambda)/d\lambda$ that appears in equation (6.9). To facilitate the averaging operation the DL_POLY Classic *utility* directory contains a program **fresta.f** which calculates the averages of the potential and kinetic energies and their RMS deviations.

6.4 Solution Spectroscopy

6.4.1 Spectroscopy and Classical Simulations

Spectroscopy is the study of the absorption of photons by an atomic or molecular species (the chromophore) to create an excited state and the subsequent de-excitation of the excited state by photon emission or quenching:

$$\begin{aligned} (Absorption) \qquad M + h\nu &\rightarrow M^* \\ (Emission) \qquad M^* &\rightarrow M + h\nu \\ (Quenching) \qquad M^* &\rightarrow M \end{aligned} \tag{6.16}$$

Clearly these are quantum mechanical processes with limited scope for modelling by classical molecular dynamics. However, if certain simplifying assumptions are made, classical simulation can yield useful information. An example of this is the calculation of solvent induced spectral shifts, in which a solvent affects a spectroscopic transition (absorption or emission) through broadening the spectral line and shifting its location in the energy spectrum. In this case the simplifying assumptions are that the spectroscopic transition occurs instantaneously without a change in the structural conformation of the chromophore (though its interaction with the solvent is expected to change as a result of the transition). In general classical simulations are concerned with the interactions between the chromophore and solvent in the ground and excited states.

DL_POLY Classic offers two capabilities in this area. Firstly, it can be used to determine the interaction energy between the solvent and the chromophore in the ground and excited states at the instant of the transition - information which quantifies the solvent induced spectral shift. Secondly, it can be used to calculate the relaxation energy resulting from the solvent response to the change in solvent-chromophore interaction after the transition.

6.4.2 Calculating Solvent Induced Spectral Shifts

A chromophore in solution differs from in vacuum by virtue of the solvent-chromophore interactions which occur in both the ground and exited states. Since the solvation energy usually different for the two states, it follows that the spectroscopic transition in solution will be different from the vacuum to an extent determined by the solvation energy difference. Spectroscopically the effect of this is to shift the location of the transition in the electromagnetic spectrum. Furthermore, the solvation energy is not constant, but fluctuates in time with a characteristic probability distribution. It follows that both the ground and excited states of the chromophore possess a distribution of possible energies which gives rise to a broadening of the spectral line.

Subject to the assumptions that the transition is instantaneous and that the chromophore retains the same geometric structure, DL_POLY Classic can calculate both of these effects. The technique is to simulate the chromophore in solution in the ground state at equilibrium and, at regular intervals, after obtaining the solvation energy of the ground state molecule, replace it with its excited state (changing its interaction potential with the solvent, but not its molecular structure) and obtain the interaction energy of the excited state. The replacement is not permanent, the excited state is used only to probe the solvation energy and has no influence on the system dynamics. It is a 'ghost' molecule. The average of the difference in the solvation energies determines the spectral shift and the distribution of the energy differences determines the line broadening.

The same procedure may be used to study the emission process. In this case the simulation is based on an equilibrated solution of the chromophore in the excited state, with replacement by the ground state chromophore at intervals.

The data produced by this option are written in the SOLVAT file (see section (6.2.3)). The *utility* programs **solsta.f** and **soldis.f** are useful for analysing these results.

6.4.3 Solvent Relaxation

Following the absorption of a photon a chromophore may persist in an excited state for an extended period. This period may be long enough for the solvent to relax around the excited chromophore and lower the configuration energy to some degree. Subject to the assumption that the chromophore structure does not change during the relaxation period the relaxation energy may be calculated by simulation. A the same time the relaxation time of the solvent may be estimated. In DL_POLY Classic this is accomplished by switching from the ground to the excited state molecule at intervals during the simulation, leaving sufficient time for the solvent to relax to equilibrium around the excited state. At the end of the chosen relaxation interval, the system may be switched back to the ground state to afford the determination of the relaxation around the ground state. The relaxation energy may be extracted from the energy difference between the equilibrated ground and excited state systems and the relaxation time from fitting to the average of many energy relaxation time plots.

The data from these simulations are written to the SOLVAT file (see section (6.2.3) at the user-defined intervals.

6.4.4 Invoking the Solvent Induced Spectral Shift Option

This option is activated by inserting the directive **excite** in the CONTROL file (4.1.1), followed by further directives to enter the control parameters and ending with the **endexc** directive. The specification is as follows.

excite
start `n1`
inter `n2`
system_a `i1 i2`
system_b `i3 i4`
endexc

The meaning of these directives is as follows.

1. **excite** - invokes the solvent induced shift option;
2. **start** `n1` - specifies the time step of the first calculation of the solvation energy of the excited state (integer `n1`);
3. **inter** `n2` - the time step interval (sampling interval) between excited state solvation calculations (integer `n2`) ;
4. **system_a** `i1 i2` identifies the range of atom indices in the CONFIG file that consitute the chromophore in the first state (integer `i1`,`i2`);
5. **system_b** `i3 i4` identifies the range of atom indices in the CONFIG file that consitute the chromophore in the second state (integer `i3`,`i4`);
6. **endexc** - closes specification of solvent induced shift option.

The following additional comments should be noted.

Atoms in the CONFIG file that are not included in the ranges defined by either directive **system_a** or **system_b** are categorised as solvent atoms. This categorisation has no effect on their physical properties. Both **system_a** and **system_b** directives specify the atoms of the chromophore, though they represent different states of the chromophore. This of course means the

chromophore appears twice in the CONFIG file. The atoms specified by **system_a** are considered to be **real** atoms and participate fully in the molecular dynamics of the system. The atoms specified by **system_b** are considered to be **virtual** and do not affect the dynamics or contribute to the energy of the system. They are however used to determine the interaction energy of the chromophore with the solvent in the manner of a virtual probe.

Important: The **system_b** atoms must be the last group of atoms listed in the CONFIG file. This is absolutely essential. (It will be necessary to restructure the FIELD file if changes are made to CONFIG.) If the chromophore is only part of a molecule instead of being the whole of it, it will be found most convenient to let the molecule containing the chromophore be the last one defined in the CONFIG and FIELD files. This will make it possible to minimise the the number of virtual atoms it is necessary to define, which reduces the file sizes and improves computational efficiency.

6.4.5 Invoking the Solvent Relaxation Option

This option is activated by inserting the directive **switch** in the CONTROL file (4.1.1), followed by further directives to enter the control parameters and ending with the **endswi** directive. The specification is as follows.

switch
start `n1`
inter `n2`
period `n3`
system_a `i1 i2`
system_b `i3 i4`
endswi

The meaning of these directives is as follows.

1. **switch** - invokes the solvent relaxation option;
2. **start** `n1` - specifies the time step at which DL_POLY Classic should first switch to the excited state (integer `n1`);
3. **inter** `n2` - the time step interval (sampling interval) between spectroscopic data calculations (integer `n2`) ;
4. **period** `n3` - the interval in time steps for the system to remain in the excited state before returning to ground state (where it will remain for an equal interval to re-equilibrate)(integer `n3`);
5. **system_a** `i1 i2` identifies the range of atom indices in the CONFIG file that consitute the chromophore in the first state (integer `i1`,`i2`);
6. **system_b** `i3 i4` identifies the range of atom indices in the CONFIG file that consitute the chromophore in the second state (integer `i3`,`i4`);
7. **endswi** - closes specification of solvent relaxation option.

See section (6.4.5) for comments on the specification of atoms in **system_a** and **system_b**, which are equally valid here. Furthermore, when **system_a** and **system_b** atoms are exchanged under the **switch** option, the former **system_a** atoms become virtual and **system_b** become real, until they are swapped over again at intervals defined by the **period** directive. The data in the SOLVAT file may be plotted to give a clear representation of the progress of the simulation and the relaxation of specific components of the solvation energy.

Chapter 7

Metadynamics

Scope of Chapter

This chapter describes the facilties within DL_POLY Classic for studying the thermodynamics of phase transitions using the method of metadynamics.

7.1 Overview

Metadynamics [67, 68] is a method for studying the thermodynamics of activated processes, for which purpose it accelerates the time scale for structural changes to occur and, at the same time, accumulates data describing the free energy surface, from which the free energy of the the structural transition may be obtained. The specific implementation within DL_POLY Classic uses this method in the context of transitions to/from crystalline phases. Phase transitions, for example from the liquid to the solid state, frequently exhibit *hysteresis* such that one state persists after it has become thermodynamically unstable. This is due to the existence of a free energy barrier between the states, which inhibits the transition. Metadynamics provides a means to overcome the free energy barrier and facilitate the phase transition on a timescale accessible by molecular dynamics simulation.

Metadynamics was originally devised by Laio and Parrinello [67] and the implementation in DL_POLY Classic is based on the methodology described by Quigley and Rodger [68]. The metadynamics routines in DL_POLY Classic were originally written by David Quigley and Mark Rodger at the University of Warwick and incorporated into the package by W. Smith.

Note that it is intended that this facility be used to study phase transitions and there is an accompanying expectation that such studies will be undertaken using either an NVT, NPT or NσT ensemble. Note also that when used together with shell model electrostatics the metadynamics routines revert to velocity Verlet integration .

7.2 Theory of Metadynamics

In metadynamics the Hamiltonian that defines the dynamics of an N-particle system is augmented by a *time dependent* bias potential which is a function of appropriate order parameters[1] that characterise the structure of the system:

$$H[\underline{r}^N, t] = \sum_{i=1}^{N} \frac{p_i}{2m_i} + U(\underline{r}^N) + V[\underline{s}^M(\underline{r}^N), t]. \tag{7.1}$$

In this equation $U(\underline{r}^N)$ is the usual potential energy function describing the interactions between, and within, the molecules. p_i is the momentum of the $i'th$ atom and m_i its mass. The novel term $V[\underline{s}(\underline{r}^N), t]$ is the time dependent bias potential, which is a function of a vector $\underline{s}^M$ that is an ordered set of M order parameters, each of which is defined by the instantaneous positions $\underline{r}^N$ of the atoms in system. The bias potential is time dependent in the sense that it can be 'grown' by adding, at periodic intervals of time τ_G, a Gaussian term of weight w and width δh:

$$V[\underline{s}^M(\underline{r}^N), t] = w \sum_{k=1}^{N_G} exp\left[\frac{-|\underline{s}^M(k\tau_G) - \underline{s}^M(t)|^2}{2\delta h^2}\right], \tag{7.2}$$

where k runs over all previously deposited Gaussians and $N_G = int(t/\tau_G)$. The force on each atom $\underline{f}_i$ derived from the Hamiltonian (7.1) is given by

$$\underline{f}_i = -\underline{\nabla}_i U(\underline{r}^N) - \sum_{j=1}^{M} \frac{\partial V}{\partial s_j} \underline{\nabla}_i s_j(\underline{r}^N) \tag{7.3}$$

If the deposition rate w/τ_G is slow enough the motion of the order parameters $\underline{s}^M$ is adiabatically separated from the motion of the atomic system. After a sufficiently long simulation, the bias potential cancels out or 'fills' the free energy landscape of the potential $U(\underline{r}^N)$ and permits an

[1] The term *collective variable* may be used as an alternative to *order parameter*.

accelerated dynamics. Meanwhile the bias potential becomes a measure of the free energy surface i.e.

$$F_G(\underline{s}^M) = -\frac{lim}{t \to \infty} V[\underline{s}^M(\underline{r}^N), t] \tag{7.4}$$

The accuracy of this estimation of the free energy surface is dependent on the deposition rate and the effective diffusion constant of the order parameters. Typically the error is of order w, the Gaussian weight. A discussion of these issues is given by Laio *et al* [69]. and Quigley and Rodger [68].

The importance of using order parameters in the Hamiltonian (7.1) is that they are a direct measure of the structure of a particular phase. Increasing the bias potential therefore has the effect of destabilising phases that are characterised by these parameters, forcing the simulation to move to alternative structures with lower free energy. In general several different order parameters can be used at the same time, to improve the control of the selectivity of the various phases and the pathways between them. However, this must be weighed against the additional computational cost, which grows exponentially with the number of order parameters.

Quigley and Rodger have described a protocol for deciding which order parameters to use in [68]. Firstly a set of simulations of the disordered state and any accessible crystalline polymorph are performed and the equilibrium distributions for the candidate order parameters obtained (see section 7.4.1). Any sets of parameters for which the distributions overlap are discarded until the sets remaining describe the known states with minimum ambiguity. This ensures that the realisable structures are distinct in the collective space of the order parameters. However this approach does not guarantee that pathways between states will match those that occur in the unbiased system, though it does set an upper bound for the corresponding free energy barrier. An alternative method devised by Peters and Trout offers a better description of pathways [70].

7.3 Order Parameters

The order parameters available in DL_POLY Classic are as follows:

1. Potential energy [71];
2. The Q4 and Q6 parameters of Steinhardt *et al* [72];
3. The ζ tetrahedral parameter of Chau and Hardwick [73];

These order parameters are described below.

7.3.1 Potential Energy as an Order Parameter

The use of potential energy as an order parameter was pioneered by Donadio *et al* [71]. It is self evident that the configuration energy is a well behaved function that takes on distinct values for different structures. It has the additional advantage that it requires no additional computation time, since it is normally calculated automatically during any molecular dynamics simulation. It is also straightforward to calculate the associated biasing forces and stress tensor contributions:

$$\underline{f}_i \rightarrow \underline{f}_i(1 + \frac{\partial V}{\partial U}) \tag{7.5}$$

$$\underline{\underline{\sigma}} \rightarrow \underline{\underline{\sigma}}(1 + \frac{\partial V}{\partial U}) \tag{7.6}$$

In addition to using potential energy as a *global* parameter Quigley and Rodger advocate its use as a *local* parameter [68], which pertains to a specific subset of atoms in the system - namely those

that form the central atoms in the definitions of the Steinhardt or tetrahedral order parameters discussed in the following sections. This allows the use of potential energy in association with these order parameters. This approach has the advantage that it allows the user to drive structural changes in parts of the system that are of greatest interest and not (say) the solvent or substrate. In implementing this, a corresponding calculation of the local potential energy and stress tensor needs to be added to each DL_POLY force routine. It turns out that it is not practical to do this in all of DL_POLY's force routines, in particular those that determine many-body forces e.g. Tersoff and metal potentials do not have this capability. A similar omission occurs with the reciprocal space term of the Ewald sum.

7.3.2 Steinhardt Order Parameters

The parameters of Steinhardt, Nelson and Ronchetti [72] employ spherical harmonics to describe the local order of atoms of type β surrounding an atom of type α, thus:

$$Q_\ell^{\alpha\beta} = \left[\frac{4\pi}{2\ell+1} \sum_{m=-\ell}^{\ell} \left| \frac{1}{N_c N_\alpha} \bar{Q}_{\ell m}^{\alpha\beta} \right|^2 \right]^{1/2} \tag{7.7}$$

where

$$\bar{Q}_{\ell m}^{\alpha\beta} = \sum_{b=1}^{N_b} f_c(r_b) Y_{\ell m}(\theta_b, \phi_b). \tag{7.8}$$

The summation in equation (7.8) runs over all N_b atoms of type β within a prescribed cutoff surrounding an atom of type α and r_b represents the scalar distance between the α and β atoms. The function $f_c(r_b)$ is a switching function that sets the cutoff range at the required separation in a continuous (and therefore differentiable) manner. It has the form:

$$f_c(r) = \begin{cases} 1 & : \quad r \le r_1 \\ \frac{1}{2}\left\{ \cos\left[\frac{\pi\ (r-r_1)}{(r_2-r_1)} \right] + 1 \right\} & : \quad r_1 < r \le r_2 \\ 0 & : \quad r > r_2 \end{cases} \tag{7.9}$$

The parameters r_1 and r_2 define a range over which the β atoms gradually cease to count towards the overall sum. Note that the numbers N_c and N_b in the above formulas are formally expected to be the same in a perfect crystal. However, while N_c remains fixed, N_b may fluctuate according to circumstance. (In fact the switching function replaces the strict cut off in the original definition by Steinhardt *et al* in which N_c would be equivalent to $\sum_{b=1}^{N_b} f_c(r_b)$ rather than a constant.) Quigley and Rodger also note that order parameter is not scale invariant between systems of different numbers of atoms [68], however this does not matter for simulation where the numbers are fixed. The spherical harmonic parameter ℓ is confined to the values 4 and 6 in the DL_POLY Classic implementation, giving the order parameters $Q_4^{\alpha\beta}$ and $Q_6^{\alpha\beta}$.

The forces arising from the Steinhardt parameters are given by:

$$\begin{aligned} \underline{f}_{ij} \quad = \quad & -\hat{r}_{ij} \frac{\partial V}{\partial Q_\ell^{\alpha\beta}} \frac{1}{Q_\ell^{\alpha\beta}} \frac{4\pi}{2\ell+1} \left(\frac{1}{N_c N_\alpha} \right)^2 \times \\ & \sum_{m=-\ell}^{\ell} \left\{ \Re(\bar{Q}_{\ell m}^{\alpha\beta}) \frac{d}{dr_{ij}} [f_c(r_{ij}) \Re(Y_{\ell m}(\theta_{ij}, \phi_{ij}))] \right. \\ & \left. + \Im(\bar{Q}_{\ell m}^{\alpha\beta}) \frac{d}{dr_{ij}} [f_c(r_{ij}) \Im(Y_{\ell m}(\theta_{ij}, \phi_{ij}))] \right\}. \end{aligned} \tag{7.10}$$

where $\Re$ and $\Im$ indicate the *Real* and *Imaginary* parts of complex quantities.

The stress tensor contributions arising from these forces are given by

$$\sigma_{\alpha\beta} \rightarrow \sigma_{\alpha\beta} - f_{ij}^{\alpha} r_{ij}^{\beta}. \tag{7.11}$$

7.3.3 Tetrahedral Order Parameters

The form of the tetrahedral order parameter in DL_POLY Classic is that of Chau and Hardwick [73], which quantify the degree to which atoms surrounding a chosen atom are arranged tetrahedrally. When the chosen atom and its surrounding neighbours are of the same type (α) the parameter is defined by formula

$$\zeta_{\alpha} = \frac{1}{N_c N_{\alpha}} \sum_{i=1}^{N_{\alpha}} \sum_{j \neq i}^{N_{\alpha}} \sum_{k>j}^{N_{\alpha}} f_c(r_{ij}) f_c(r_{ik}) (cos\theta_{jik} + 1/3)^2, \tag{7.12}$$

where indices i, j and k run up to N_{α} atoms of type α. Integer N_c and function $f_c(r)$ are as for the Steinhardt parameters (i.e. f_c once again replaces a fixed cut off and N_c is a fixed constant). This order parameter is maximal for tetrahedral atomic arrangements. The atomic forces that arise from this order parameter can be expressed in terms of pair forces between atoms i and j and between i and k which are given by

$$\begin{aligned} \underline{f}_{ij} &= -\frac{\partial V}{\partial \zeta_{\alpha}} \left\{ \frac{2}{r_{ij}} f_c(r_{ij}) f_c(r_{ik}) (cos\theta_{jik} + 1/3)(\hat{r}_{ik} - \hat{r}_{ij} cos\theta_{jik}) \right. \\ &\quad \left. + (cos\theta_{jik} + 1/3)^2 \frac{df_c(r_{ij})}{dr_{ij}} f_c(r_{ik}) \hat{r}_{ij} \right\} \end{aligned} \tag{7.13}$$

$$\begin{aligned} \underline{f}_{ik} &= -\frac{\partial V}{\partial \zeta_{\alpha}} \left\{ \frac{2}{r_{ik}} f_c(r_{ij}) f_c(r_{ik}) (cos\theta_{jik} + 1/3)(\hat{r}_{ij} - \hat{r}_{ik} cos\theta_{jik}) \right. \\ &\quad \left. + (cos\theta_{jik} + 1/3)^2 \frac{df_c(r_{ik})}{dr_{ik}} f_c(r_{ij}) \hat{r}_{ik} \right\}. \end{aligned} \tag{7.14}$$

The stress tensor contributions can be described in terms of these forces:

$$\sigma_{\alpha\beta} \rightarrow \sigma_{\alpha\beta} - f_{ik}^{\alpha} r_{ik}^{\beta} - f_{ij}^{\alpha} r_{ij}^{\beta}. \tag{7.15}$$

7.3.4 Order Parameter Scaling

The order parameter vector $\underline{s}^M$ consists of an ordered set of different order parameters and it is not generally the case that all of them return numbers of the same order of magnitude. This is particularly true for the potential energy. It is therefore sensible that when using the order parameters collectively to define the state of a system, that they should be scaled to give numbers of simular magnitudes. So when specifying order parameters to define the metadynamics DL_POLY Classic allows the user to include a scale factor in the definition. This appears in the STEINHARDT and ZETA data files described in the following section.

7.4 Running Metadynamics Simulations

The recommended procedure for running metadynamics with DL_POLY Classic is as follows.

1. Scope out the appropriate choices of order parameters for your system following the method of Quigley and Rodger outlined above in section (7.2) and in [68]. You should use one of the (equilibrated) REVCON files as the starting configuration for your metadynamics study.

2. Decide a suitable interval τ_G for depositing the Gaussians (7.2) and an appropriate Gaussian height w and width δh to ensure accuracy for the free energy calculation. See [69] and [68] for details. Along with this the user must also choose a Gaussian deposition/convergence scheme (see section 7.4.1).

3. Set up the metadynamics option in the CONTROL file as follows:

   (a) Set the metadynamics directive:
   metadynamics
   Then enter the metadynamics control variables one per record as indicated below. Comment records may be inserted if the first character is the hash symbol (#) or the ampersand (&).

   (b) Set the number of order parameters to be used (*ncolvar*):
   ncolvar n
   where n is an integer.

   (c) If Steinhardt order parameters are required, activate the option with the directive:
   lstein

   (d) If tetrahedral order parameters are required, activate the option with the directive:
   ltet

   (e) If the global potential energy order parameter is required, activate the the option with the directive:
   lglobpe

   (f) If local potential energy order parameters are required, activate the option with the directive (note that global and local potential energy are mutually exclusive options):
   llocpe

   (g) Set the scale factor for the global potential energy order parameter:
   globpe_scale f
   where f is a real number.

   (h) Set the scale factor for the local potential energy order parameter:
   locpe_scale f
   where f is a real number.

   (i) Set the number of Steinhardt Q_4 parameters required:
   nq4 n
   where n is an integer.

   (j) Set the number of Steinhardt Q_6 parameters required:
   nq6 n
   where n is an integer.

   (k) Set the number of tetrahedral ζ parameters required:
   ntet n
   where n is an integer.

   (l) Set the Gaussian potential deposition interval (in units of time steps):
   meta_step_int n
   where n is an integer.

   (m) Set the height (w) of the Gaussian potentials (in units of k_BT, where T is the simulation temperature):
   ref_W_aug f
   where f is a real number.

(n) Set the Gaussian potential width parameter:
h_aug f
where f is a real number.

(o) Set the Gaussian control key:
hkey n
where n is an integer. See section (7.4.1) for guidance.

(p) Set the control parameter for well-tempered dynamics (required when **hkey** is 2):
wt_Dt f
where f is a real number.

4. Set other CONTROL file directives as follow:

(a) Select the **restart noscale** option if the CONFIG file was pre-equilibrated, otherwise leave out the **restart** keyword altogether.

(b) Set the length of the simulation required (**steps**) and the equilibration period (**equil**) (both in time steps). The equilibration can be short or absent if the system was pre-equilibrated.

(c) You **must** select one of the Nosé-Hoover NVT, NPT or NσT ensembles. Metadynamics is only available for one of these options. The program automatically defaults to velocity Verlet integration, if you use shell model electrostatics.

(d) If you wish to follow the structural changes, set the **trajectory** option in the CONTROL file. This will produce a HISTORY file you can view or analyse later.

(e) Set the remaining CONTROL keywords as for a normal molecular dynamics simulation.

5. Prepare, if required, the file STEINHARDT, which defines the control variables for the Steinhardt order parameters. The file specification is as follows.

(a) The file contains data for both Q_4 and Q_6 order parameters.

(b) The records describing the Q_4 entries appear first. There is one information record followed by $nq4$ data records. ($nq4$ is defined in item 3(i) above.) No Q_4 entries appear if $nq4$ is zero. After this, the records for the Q_6 parameters appear. There is one information record, followed by $nq6$ data records. ($nq6$ is defined in item 3(j) above.) The records are free format and the content of the information records is ignored by the program.

(c) Each data record (for both Q_4 and Q_6) consists of (in order)
- The name of the atom type α (max. 8 characters);
- The name of the atom type β (max. 8 characters);
- The control parameter r_1 for the function $f_c(r)$ in equation (7.9) (real);
- The control parameter r_2 for the function $f_c(r)$ in equation (7.9) (real);
- The scale factor for the order parameter (real);
- The number N_c of expected atoms of type β around the α atom (integer).

6. Prepare, if required, the file ZETA, which defines the control variables for the tetrahedral order parameters. The file specification is as follows.

(a) The file consists of 1 information record followed by $ntet$ data records. The data records are free format and the content of the information record is ignored by the program.

(b) Each data record consists of (in order)
- The name of the atom type α (max. 8 characters);
- The cutoff parameter r_1 for the function $f_c(r)$ (real);
- The cutoff parameter r_2 for the function $f_c(r)$ (real);
- The scale factor for the order parameter (real);
- The number N_c of expected atoms of type α around each α atom (integer).

7. Run the metadynamics simulation. This will perform a simulation at the temperature and pressure requested. When the simulation ends, proceed as follows.

(a) Check the OUTPUT file. Make sure the simulation behaved sensibly and terminated cleanly. If the required number of time steps has not been reached, the simulation can be restarted from the REVCON and REVIVE files (renaming them as CONFIG and REVOLD for the purpose) and setting the directive **restart** (with no qualifier) in the CONTROL file.

(b) Use the DL_POLY Java GUI to plot the system energy and temperature for the whole of the simulation and make sure no key variables misbehave.

(c) The HISTORY file, if requested, contains the trajectory of the system. This may be viewed, or otherwise investigated, by appropriate viewing or analysis software.

(d) The METADYNAMICS file produced by the run contains the data describing the evolution of the time dependent bias potential (equation 7.1). This file consists of a series of records, the content of each is:
- The *meta-step* (current time step number divided by the Gaussian deposition interval *meta_step_int*). Format (i8).
- All *ncolvar* order parameters selected. Format *ncolvar*(e15.6).
- The height of the deposited Gaussian divided by k_bT. Format (e15.6).

7.4.1 Additional Considerations

1. **Choosing the Gaussian convergence scheme**. DL_POLY Classic offers three different schemes for handling the addition of the Gaussians to the bias potential. Further details on these schemes can be obtained from Quigley and Rodger [68].

(a) **Simple addition**. Gaussians with fixed height and width parameters are simply added to the bias potential. This option is selected by setting **hkey=0** in the metadynamics section of the CONTROL file.

(b) **Wang-Landau recursion**. Starting from a given Gaussian height a histogram is accumulated with each Gaussian addition recording the visits to each square of the order parameter space. Once this histogram is approximately (say 80%) flat, the Gaussian height is halved, the histogram is reset to zero and then the process continues until a higher degree of flatness has been achieved, and so on. The procedure is meant to ensure that the added Gaussians make progressively finer contributions as convergence is approached. Set **hkey=1** for this option. (Note this option has been implemented only for the case where **ncolvar=1**.)

(c) **Well-tempered dynamics**. The well-tempered scheme uses a maximum energy criterion. A threshold energy V_{max} is set above the largest expected energy barrier and at each step the Gaussian deposition height is given by

$$w = w_0 exp[-V_{aug}(\underline{s}^M)/V_{max}]$$

where V_{aug} is the current value of the bias energy. The parameters w_0 and V_{max} are defined by the input directives **ref_W_aug** and **wt_Dt** in the CONTROL file (see above). Set **hkey=2** for this option.

2. **Defining the switching function** $f_c(r)$
The switching function is determined by the parameters r_1 and r_2 in formula (7.9). These must be chosen so that r_2 absolutely excludes near-neighbouring atoms that are not intended to be considered part of the sum in equation (7.7) or (7.12). r_1 should not be so short that is sometimes does not include atoms that should be fully counted. The range $r_1 \rightarrow r_2$ should be set to correspond to the minimum in the appropriate pair correlation functions in the relevant system states. This choice minimises spurious forces that can arise from order parameters that have different ranges [68].

3. **Using DL_POLY Classic to help select appropriate order parameters**
Section (7.2) outlined a protocol for deciding suitable order parameters for a particular metadynamics study, which required the construction of distribution functions for candidate order parameters. These may be obtained from DL_POLY Classic by perform simulations of the relevant system states with the **metadynamics** directive set in the CONTROL file, but with the Gaussian accumulation disabled by setting the Gaussian height parameter **ref_W_aug** to zero. The resultant dynamics will be *time independent* and the METADYNAMICS file will tabulate values of the order parameters at regular intervals. It remains then to construct histograms of these parameters to determine the degree of overlap between them as required.

4. **Choosing order parameter scaling factors**
The widths of the histograms for the order parameter distributions described in the previous paragraph should also be used to set the appropriate order parameter scaling factors referred to in section (7.3.4).

5. **Deciding the simulation length**
Deciding that the metadynamics simulation has been long enough is a matter of judging whether the system is diffusing like a random walk in the space of the order parameters i.e. that it is sampling all the available parameter space. This is somewhat easier in the cases of Wang-Landau recursion and well-tempered dynamics as it is indicated by the parameter w (the Gaussian height) becoming relatively small. In general a degree of experience in the technique is required to make a good judgement.

6. **Contra-indications**
A useful point to note is that if a simulation does not reach a state where transitions between minima occur rapidly without residual hysteresis, then by implication the original choice of order parameters was poor.

7.4.2 Analysing the Metadynamics Results

Analysis of the results of a metadynamics simulation can take a number of forms, some of which are outlined here.

1. **Determination of free energy**
The information in the METADYNAMCS file is sufficient to define the system free energy surface through equation 7.4. The free energy is a function of the M order parameters in the vector $s^M(\underline{r}^N)$. This information can be used to determine the free energy of activation and free energy differences between states in the following manner. Firstly the free energy is projected down to a smaller subset of order parameters (usually about 2) by integrating

$exp(-F/k_BT)$ over the other order parameters and then Boltzman-inverting. (This is an expensive operation if $M > 3$). Once the free energy is mapped onto fewer dimensions the free energy barrier heights and free energy differences can be read off directly.

2. **Following the system trajectory**
It is often useful to track the trajectory of the system in the space of the order parameters to see how well the simulation is exploring that space. For this purpose is is possible to plot the contents of the METADYNAMICS file graphically in a selection of 2D sections. Simple graphics are generally sufficient for this purpose. Alternatively, the HISTORY file may be viewed as a movie, using packages such as VMD [74] to show the transformations that occur.

Chapter 8

The Java Graphical User Interface

Scope of Chapter

This chapter describes in detail the functionality, compilation and operation of the Java GUI.

8.1 The DL_POLY Java Graphical User Interface

The DL_POLY Java GUI II is an upgrade of the original (Mark I) version of the GUI and incorporates several new features. First among these is the Graphical Molecular Editor, which allows the user to build complex organic structures and replicate them to create systems for simulation with DL_POLY. To accomplish this the GUI has two graphical modes:

- View - for simply viewing the molecular structure in a DL_POLY CONFIG file and also carrying out certain global operations (i.e. affecting the whole configuration) such as insertion of water molecules, replication etc. Which do not change the contents of the CONFIG file as such, but augment them in some way. This mode is equivalent to the previous, sole mode of operation of the Mark I version of the GUI.
- Edit - for graphically editing the structure of a configuration, changing the identities of molecules or atoms, adding or deleting molecules, parts of molecules or atoms.

Secondly, improvements have been made to the force field builders, which now include several new Java classes related to system structures and molecular entities. This will hopefully allow incorporation of additional force fields more easily than before.

The GUI retains the advantages inherent in the previous version, particularly in the free availability of Java, a product of Sun Microsystems, and its portability to all computers that support the Java language. We continue to supply the code as source (though the executable is also included) so it may be developed as desired by users with special needs, provided they are of course willing to learn the Java language.

Other aspects of the new GUI the user should be aware of are as follows:

1. **The DL_POLY Java GUI is not an applet. It is a full Java application program and cannot be run from within an internet browser.**
2. **The appearance of the GUI, its windows and browsers etc may vary from system to system. The broad appearance will remain the same as will the functionality. This is merely an aspect of Java portability.**
3. **For the purpose of rendering different atom types in a configuration, the GUI requires the adoption of a naming convention based on the Periodic Table. Up to eight characters may specify an atom name, but the first two must be taken from the proper chemical symbol. If the chemical symbol is a single letter, the underscore (_) must be used. Thus hydrogen and oxygen have the names O_ and H_, while copper and sodium have Cu and Na respectively. Failure to follow this convention results in the 'grey ball' syndrome. The only exceptions to these rules are the use of symbols OW and HW for oxygen and hydrogen in water molecules. Other atom naming conventions can be fitted into this scheme by extension beyond two characters. The GUI exploits this in the Dreiding [7], OPLS [22] and Ceramics [75, 76] force fields. The user may see what the atom names are in these force fields by inspecting the MINDREI, MINIOPLS and CERAMICS files in the java subdirectory, or by listing them from the Information menu of the GUI (see section 8.11).**

8.2 Compiling the Java GUI

The first requirement for compiling the Java GUI is the availability of the Java software. For national supercomputers and the like, Java should already be available. To install Java on local

machines, the Java home page at the Sun Microsystems website http://java.sun.com provides the Java Development Kit (JDK 1.4 or above) for any particular computer. Installation instructions are available from the same site, and are generally straightforward. Once the JDK is installed, the DL_POLY Java GUI may be compiled.

The source code for the Java GUI is found in the DL_POLY java subdirectory. On entering this type the command:

*javac *.java*

which will compile the java source code and construct the Java classes (i.e. the Java executable objects). Next it is necessary to make a Java jar file from the classes. This neatly encapsulates, in a single file, all of the GUI Java classes. (The jar file effectively becomes the GUI executable, which in fact is transportable between systems.) This is done with the command:

*jar -cfm GUI.jar manifesto *.class About_DL_POLY Acknowledge CERAMICS MINIDREI MINIOPLS WATER.300K TestInfo Licence Disclaimer*

the jar command thus works somewhat like the UNIX tar command. Note particularly the requirement to incorporate the file "manifesto", which is one of the files in the java subdirectory. The contents of this file inform the java program which of the incorporated classes represents the entry point at execution. (It follows that this file should never be deleted!) The files listed after the *.class inclusion are information the GUI requires to operate properly. Note that both the javac and jar commands have been combined in the unix script 'build, which you will find in the java directory. Invoke this by typing the command

./build

The result of this command is the GUI.jar file, which becomes the working GUI. In addition to compiling the Java source, it is necessary to compile some FORTRAN programs that the GUI calls upon to perform some of the calculations. Inside the java subdirectory are the lower directories GDF and SKW. These contain programs for calculating van Hove correlation functions and dynamic structure factors respectively. Enter each of these directories and type 'make to build the FORTRAN programs. This will complete the build of the GUI. Note it may be necessary to change a few of the parameters in the makefiles for these, such as the specification of the FORTRAN77 compiler or the C compiler.

8.3 Starting the Java GUI

The GUI.jar file is universally transportable. Once created it can be shipped to any system where Java is featured and it will run just the same (though the various windows may look a little different). In the case of MS Windows machines, it can be activated by double-clicking on the file icon. It is normal practice to run the GUI from within the DL_POLY execute subdirectory, which makes it easy to locate the files it needs from within the DL_POLY directory structure. (Users who experience strange problems running the GUI may find that reverting to this practice resolves the problems.) Assuming you are in the execute directory, to start the GUI in a linux or MS Windows command window type the command:

java -jar ../java/GUI.jar

Unix users may alternatively use the gui script in the execute subdirectory, which incorporates the syntax of this command. After a minute or two, the GUI Graphics Window and the Monitor Window (figure 1) will open onscreen. Both of these windows may be handled in the usual X manner, e.g. moved with the mouse, by clicking and dragging the header panel. The size of each

may be changed by clicking and dragging from a corner or edge. The usual window widgets for hiding, enlarging or closing are present. However note that the close widget has been disabled for the Monitor Window. The GUI opens in the View mode.

Figure 1: The Java GUI and Monitor window

Both of these windows may be handled in the usual X manner, e.g. moved with the mouse, by clicking and dragging the header panel. The size of each may be changed by clicking and dragging from a corner or edge. The usual window widgets for hiding, enlarging or closing are present. However **note that the close widget has been disabled for all Java GUI windows except the Graphics Window**.

If the colour scheme of the GUI offends, try the command:

java -jar ../java/GUI.jar scheme

where *scheme* is one of the following: **photo, monet, picasso, vangogh, cezanne** or **mondrian**, any of which may be more pleasing. The default colour scheme is **picasso**.

8.4 The Graphics Window

The Graphics window is the area in which the GUI draws pictures of any molecular configuration selected by the user, or produced by the GUI. The view represents the x,z plane, with the x-axis horizontal, z-axis vertical and the positive y-axis projecting into the screen. On the right of this

window is a collection of buttons, which manipulate the image on display. These are activated by a single click of the mouse button. Their functions are as follows:

- **New** - opens a file browser (see figure 2) for the user to select a new configuration file for viewing. This button assumes the configuration file is a DL_POLY CONFIG or REVCON file.
- **Cls** - clears the image from the screen and deletes the configuration data from the GUI internal memory.
- **Rst** - resets the image to the original picture when the configuration was first loaded. i.e. undoes all user manipulation with the GUI.
- **Prn** - throws up a print panel so the user may either print the image or save the image as a file.
- **Tx**− - moves (translates) the image to the left.
- **Tx**+ - moves the image to the right.
- **Ty**− - zooms in on the image.
- **Ty**+ - zooms out from the image.
- **Tz**− - moves the image down.
- **Tz**+ - moves the image up.
- **Rot** - rotates the image to follow the dragged cursor.
- **Tra** - moves the image to follow the dragged cursor.
- **Rx**− - rotates the image clockwise about the x axis.
- **Rx**+ - rotates the image anticlockwise about the x axis.
- **Ry**− - rotates the image clockwise about the y axis.
- **Ry**+ - rotates the image anticlockwise about the y axis.
- **Rz**− - rotates the image clockwise about the z axis.
- **Rz**+ - rotates the image anticlockwise about the z axis.
- **H2O** - toggles the visibility of water molecules.
- **Bnd** - toggles the visibility of stick bonds.

Note that toggling the visibility of water molecules assumes that the oxygen atom is labelled as OW and the hydrogen atoms as HW in the CONFIG file. Clicking the **Edt** button in View mode activates another column of buttons (and also changes slightly the function of some of the buttons described above). The function of these will be given in the section on the Molecular Editor, which appears later in this document.

8.5 The Monitor Window

The Monitor window is the medium through which the GUI informs the user of actions taken, files created or deleted, errors made by the user etc. It is also the area where text files are displayed if the user requests it. Ideally this window should be kept visible at all times. It is expandable if required and can be reduced (hidden) but not deleted.

Note that if there are catastrophic failures during the running of the GUI, the error messages will, of necessity, not appear in this window, but in the X or command window in which the GUI was first invoked.

8.6 The GUI Application Menus

Also appearing in the Graphics window (top left) is a menu bar with are a series of drop down menus (made visible by clicking on the menu name). These menus select the various applications buried in the GUI. The applications are discussed in detail below. The current menus are:

- **File** - handles file operations, such as viewing and deleting, resets the GUI and various defaults, and also quits (shuts down) the GUI.
- **FileMaker** - enables construction of various input files for DL_POLY.
- **Execute** - controls the execution of DL_POLY and the selection and storage of I/O files.
- **Analysis** - runs the DL_POLY analysis programs to analyse the simulation results.
- **Information** - provides licencing and other information.

The contents of each menu are made visible by clicking on the menu header.

8.7 File Menu

The File menu contains the following items

1. **Quit**
2. **View File**
3. **Delete File**
4. **Defaults**
5. **Reset**

To select any item from the menu, the mouse cursor must be dragged down the list and released on the item of choice. Selecting any of these items will initiate a particular action by the GUI. These actions are now described.

8.7.1 Quit

Selection of this item closes down the GUI, provided it is not in a busy state (i.e. performing some other action). No data saving results from this action. Amendments to the GUI defaults will be lost. The GUI may also be shut down by clicking on the standard close widget on the GUI window.

8.7.2 View File

Any text file in the *execute* subdirectory may be viewed by selecting this item. On selection the GUI opens a file browser (figure 2) allowing the user to select the file of interest. Selection of the file will result in its display in the Monitor window (which may need to be enlarged for a convenient viewing).

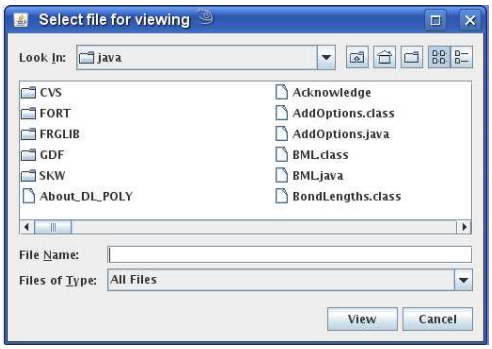

Figure 2: A typical file browser

Note that only text files should be viewed, for obvious reasons. Large files may take quite a while to list.

8.7.3 Delete File

Any file in the *execute* subdirectory may be deleted by selecting this item. On selection a file browser appears to permit the user to choose the required file. Selection of the file will cause its deletion. As a fail safe, the GUI will first throw up a dialog window to give opportunity to cancel the action.

8.7.4 Defaults

The GUI contains some defaults that may be altered after start up. These include array dimension specifications and GUI operational parameters, such as default rotation angles and translation distances. Selection of this item opens the Change Defaults panel (figure 3).

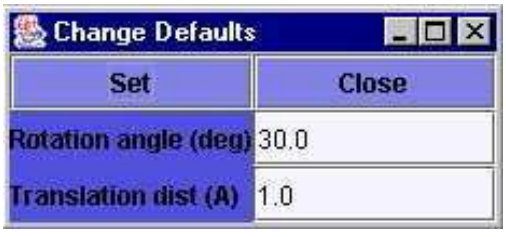

Figure 3: The Change Defaults Panel

The text boxes on the panel show the current values for the defaults. Clicking on the text box beside the parameter description will allow the user to change the value. Clicking the **Set** button

will reset the GUI default. Sometimes this also resets the entire GUI, if necessary. Clicking the **Close** button will delete the panel.

The parameters that may be changed are:

- Rotation angle (deg) - angle of rotation for Graphics window rotation buttons.
- Translation dist (A) - distance of translation for Graphics window translation buttons.

8.7.5 Reset

This action resets the GUI variables back to their starting values (not current defaults) and clears all loaded data from the GUI.

8.8 FileMaker Menu

The FileMaker menu has the following items, some of which are themselves submenus.

1. **CONTROL**
2. **CONFIG**
3. **FIELD**
4. **Display**
5. **Tools**

The functions of these are as follows.

8.8.1 CONTROL

Selecting this menu item will open the first of three panels dedicated to constructing a DL_POLY CONTROL file. A second panel is opened by clicking the **More** button at the bottom of this, and a third panel is opened by clicking the **More** button at the bottom of the second panel. The three panels (figure 4) allow specification of all the options available in DL_POLY, as specified in the DL_POLY manual, section 4.1.1. The details of each item do not require elaboration here.

Specifying most values required by the CONTROL file is matter of amending the contents of the associated text box. Some variables however, require activation of logical options using 'check boxes' (particularly panel 3). Yet others require a choice from a menu of options. The choice of ensemble or electrostatic method on panel 1 or the choice of restart option on panel 2 are examples of this. The user will find the panels intuitively straightforward.

When the variables have been defined, the user should click the **Make** button on panel 1, to create the required CONTROL file. The GUI will name this file CNTROL.n, where n is some integer. Note that the GUI has some internal consistency checks and may refuse to make a CONTROL file that is improperly specified. Watch the Monitor window for details.

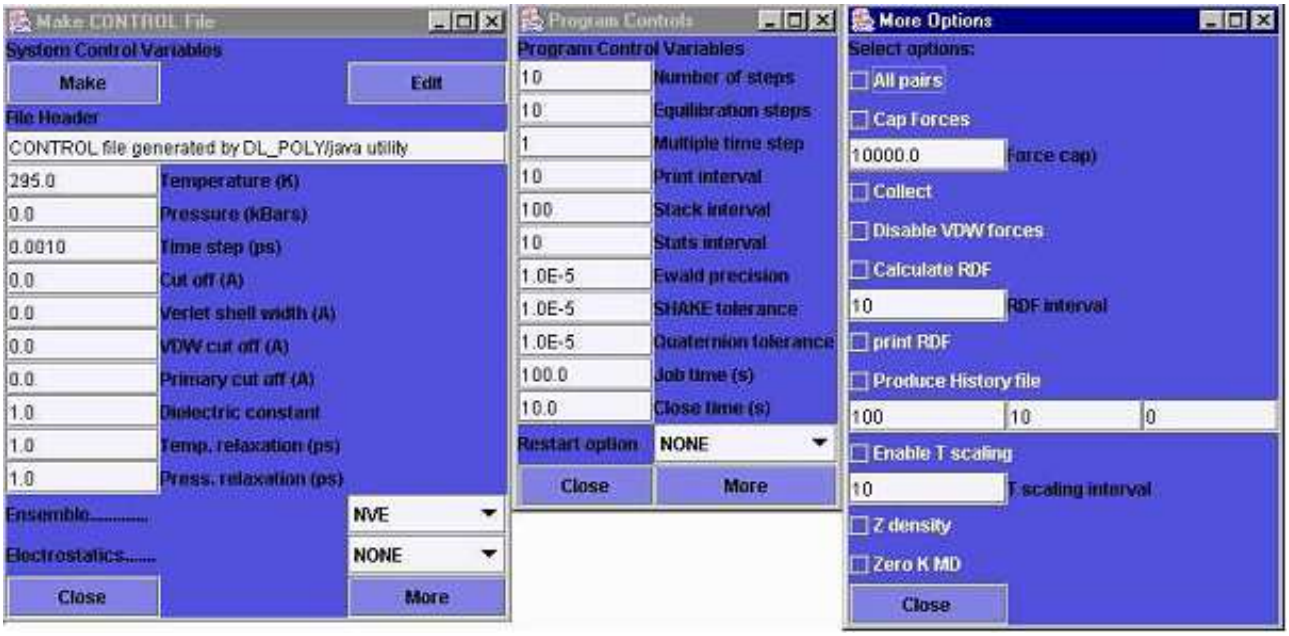

Figure 4: The CONTROL file panels

Another way of making a CONTROL file is to use the **Edit** button on panel 1 to load an existing CONFIG file. This may them be altered in some way through the panels and saved using the **Make** button. A new CONTROL file will be made alongside the old.

When all is finished, the CONTROL file panels may be deleted using the close button on panel 1. Note you cannot open more than one set of CONTROL panels. This is a restriction that applies to most other panels of the GUI, otherwise things could get confusing.

8.8.2 CONFIG

This menu item is intended for building DL_POLY CONFIG files. It presents a submenu with the following items

1. **Lattice**
   This item is used for building CONFIG files that may be constructed from a small unit cell, such as a crystal structure. A panel is opened (figure 5) by this option which enables full specification of the CONFIG file contents.

   On the panel are several text boxes grouped in threes. The first of these specifies the unit cell A vector, the second the B vector and the third, the C vector. The distances are expressed in Å. Beneath these is a set of three boxes to specify the integer replication factor for the unit cell in each of the principal directions ($n_a \times n_b \times n_c$). The atomic basis for the unit cell is defined at the bottom of the panel. For each atom in the unit cell an atom name may be entered in the text box, followed by the three fractional coordinates of this atom in the unit cell, The **Enter** button will register the specified atom, with an accumulation number displayed on the panel, bottom right. (Note: It is not essential to use the GUI naming conventions here, but it is wise to do so, especially if one of the recognised GUI force fields is to be used later.)

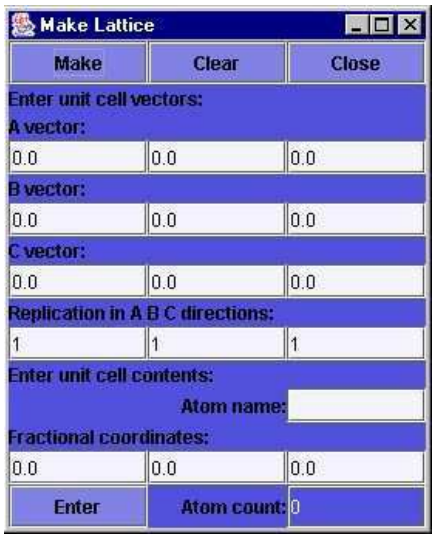

Figure 5: The Make Lattice Panel

When all the required basis atoms have been entered, the user should click the **Make** button to create the desired CONFIG file, which the GUI will name CFGLAT.n, with n some integer. Simultaneous with this, the lattice is displayed in the Graphics window. The panel can be reset by using the **Clear** button.

The **Close** button deletes the 'Make Lattice' panel.

2. **Chain**
   This panel invoked by this option (figure 6) enables construction of a small range of chain molecules, particularly surfactants, which may then form the basis for a layered system.

   The panel supports several labelled text boxes, with which the user may specify the number of carbon atoms in the chain and the XY area per chain(in $\mathring{A}^2$), as required for a layer definition. (A hexagonal arrangement of the chains is assumed.) The chain may be assigned a head group from the menu box on the panel. The current choices are:

   (a) none;
   (b) soap i.e. -CO.ONa;
   (c) carboxy i.e. -CO.OH;
   (d) phenol i.e. -C_6H_4OH;
   (e) TAB, trimethylammino bromide i.e. (-$N(CH_3)_3^+$.Br^-;
   (f) $(EO)_n$, polyethylether i.e. $(-C_2H_4O-)_n$.

   The number n in the $(EO)_n$ case is specified in a labelled text box on the panel. There are also two check boxes. One activates the 'flip' option, which turns the chain through 180 degrees, the other duplicates the chain and stacks the two chains one above the other in the z-direction, as in a double layer. The separation between the two is determined by the value in the 'Z-gap' text box. When the user has selected the required details, clicking the **Make** button will create the CFGCHN.i file, where 'i' is some integer, and simultaneously display the molecule in the Graphics window. Note that the atom naming convention adopted by this facility is compatible with the DL_POLY and Dreiding conventions.

Figure 6: The Make Chain Panel

Clicking the **Close** button will delete the 'Make Chain' panel.

Note that in order to build a full layered system, the user should use the N_fold option that appears under the FileMaker/Tools menu.

3. **Polymer**

The polymer panel opened by this option (figure 7) provides a means to construct an amorphous polymer chain by a temperature dependent self avoiding random walk.

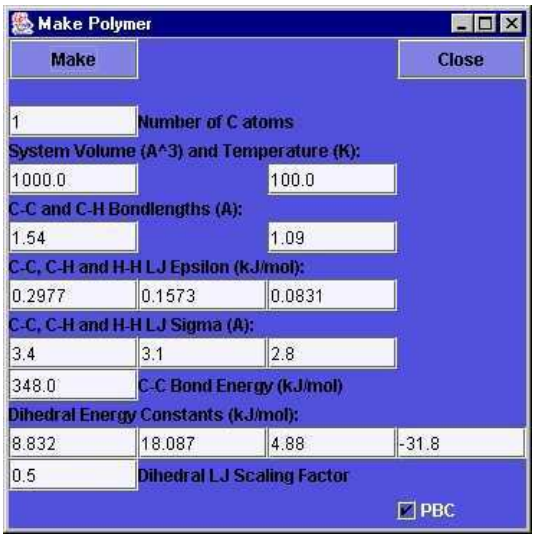

Figure 7: The Make Polymer Panel

This is a rather complicated panel, with many adjustable parameters, but the user need not know too much detail to use it. The first requirement is to specify the desired length of the chain in the first text box. The user needs then to specify the required system. volume and temperature in the appropriate labelled text boxes. Clicking the **Make** button will start the Monte Carlo process that generates the chain. If this is successful a CONFIG file named

CFGPOL.n, where n is some integer, will be written. The atom naming convention adopted by this facility is compatible with the DL_POLY and Dreiding conventions.

Other labelled text boxes on the panel allow the user to change the particulars of the model. The bondlengths, Lennard-Jones parameters (ϵ, σ), the C-C bond energy, and the (third order cosine polynomial) dihedral potential parameters are all modifiable, as is the strength of the dihedral 1-4 Lennard-Jones term, which may be scaled by a specified factor.

The chain is always grown with a cubic periodic boundary condition applied. The volume of the box is that specified by the user. However, selecting the PBC check box on the panel, will result in the chain coordinates being written into the CFGPOL file in a 'folded' manner, as opposed to a contiguous, unbroken chain.

The chain build may not be successful, depending on the temperature and volume specified by the user. If unsuccessful the user is advised to increase the volume until success is achieved. The required density may always be obtained in a DL_POLY simulation using one of the NPT ensemble options.

The **Close** button deletes the 'Make Polymer' panel.

4. **Bucky**
   The panel for this option (figure 8) enables the building of a buckminsterfullerene (C60) molecule or a nanotube.

Figure 8: The Make Fullerene Panel

To make a C60 molecule the user need only click the **C60** button on the panel. This will create a CONFIG file named CFGBUK.n, with n some integer, and simultaneously display the molecule in the Graphics window. The C-C bondlength may be set to a new value using the labelled text box. To build a nanotube, the circumference and length of the required tube must first be specified. These are given in terms of the number of C_6 rings in each direction. Two text boxes are available for this, the labelled X box refers to the circumference, and the Y box to the length. (Note that an odd number of rings specified in the Y box, will produce a continuous tube in the system z direction, by virtue of the periodic boundary condition. An even number will be incommensurate with the PBC.) Clicking the **Tube** button will create the CONFIG file CFGBUK.n and display the structure in the Graphics window.

The **Close** button will delete the 'Make Fullerene' panel.

Note that the atom naming convention adopted by this facility is compatible with the DL_POLY and Dreiding conventions.

8.8.3 FIELD

This menu item is for building DL_POLY FIELD files. It opens a submenu with the following items;

1. **Blank**
   The Blank FIELD panel (figure 9) is useful for analysing the molecular topology for a CONFIG file, since it will identify the atoms, bonds, angles, dihedrals and inversions in the system and list them in a form compatible with the DL_POLY FIELD specification. However it does not assign force field parameters.

Figure 9: The Blank Field Panel

   Operation of this panel is simple. The user needs only to click the **Make** button (if the required molecule is already loaded into the GUI), or the **Load** button (to throw up a file browser to select the required CONFIG file), to produce the blank FIELD file, which will have the name FLDuvw.n, where uvw is a three character string derived from the input CONFIG file name and n is an integer. If the **Load** button is used, the user can display the loaded file by activating the 'Display cfg' option.

   The **Close** button deletes the 'Blank Field' panel.

   Note that for the Blank FIELD facility to work properly, atomic names in the CONFIG file must obey the GUI atomic naming convention (see 3).

2. **Dreiding**
   For CONFIG files that are specified with the Dreiding naming convention [7] (see also the MINIDREI file details under the Information menu described below), the panel for this option (figure 10) may be used to build a compatible FIELD file.

Figure 10: The Dreiding FIELD Maker Panel

This is also a simple panel to operate. A FIELD file may be built from a pre-loaded CONFIG file by clicking the **Make** button, or by clicking the **Load** button to load a file from a browser. The file created will have the name FLDuvw.n , where uvw is a three character string taken from the loaded CONFIG file name and n is an integer. If the **Load** option is taken, the user may display the loaded structure by setting the 'Display cfg.' check box. The nature of the bonding forces may be altered by selecting rigid bonds or *Harmonic* or *Morse* bond potentials, and by choosing either *Len-Jones* or *Buckingham* nonbonded potentials from the box menus. The 'Use charges' check box enables the GUI to pick up prescribed charges for some atoms (such as the chloride ion). In general however this is not very useful, as the charges for the rest of the molecule (if any) have to be obtained from elsewhere.

The **Close** button deletes the 'Dreiding FIELD Maker' panel.

3. **OPLS**
   The OPLS Field File maker can create an OPLS compatible FIELD file for approriate systems generated by the GUI, or systems compatible with the CONFIG file format with the naming conventions of either OPLS or Dreiding. Selecting this menu option will result in the appearance of the 'OPLS FIELD Maker' panel (figure 11). With this panel an OPLS FIELD file may be built from a pre-loaded CONFIG file by clicking the Make button, or by clicking the Load button to load a file from a browser. The FIELD file will have the name FLDOPL.n, where n is an integer. Since the OPLS force field is a united atom force field, this option will edit out redundant hydrogen atoms and recreate the CONFIG file. The new file will be named CFGOPL.n, where n is an integer. The Close button deletes the OPLS FIELD Maker panel.

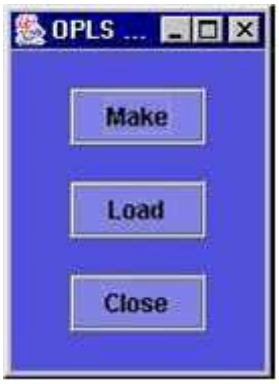

Figure 11: The OPLS FIELD Maker Panel

4. **Ceramics**
   For CONFIG files that are specified with the naming conventions for the Ceramics force fields (see the CERAMICS file details under the Information menu described below), the panel for this option (figure 12) will build an appropriate FIELD file.

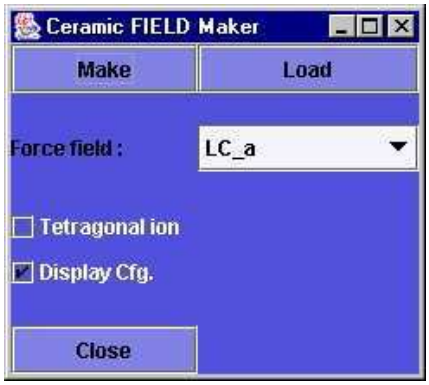

Figure 12: The Ceramics FIELD Maker Panel

With this panel a ceramic FIELD file may be built from a pre-loaded CONFIG file by clicking the **Make** button, or by clicking the **Load** button to load a file from a browser. The FIELD file will have the name FLDuvw.n, where uvw is a three character string extracted from the loaded CONFIG file name and n is an integer. The user may display the loaded structure, if not already visible, by selecting the 'Display cfg.' check box. Five ceramic force fields are available:

(a) LC_a, divalent and tetravalent rigid ions [75];
(b) LC_b, trivalent rigid ions [75];
(c) LC_c1, shell model ions [75];
(d) LC_c2, shell model ions [75];
(e) GULP, shell model ions [76];

The user may choose any of these from the menu on the panel. The LC_c1 force field distinguishes between ions in tetrahedral and other environments, for which different parameters are available. The labelled check box enables the user to specify the tetrahedral option. The **Close** button deletes the 'Ceramics Field Maker' panel.

5. **Table**
   DL_POLY allows the user to specify nonbonded pair potentials in tabular form. The 'Make TABLE File' panel (figure 13) offers some facilities for making such files.

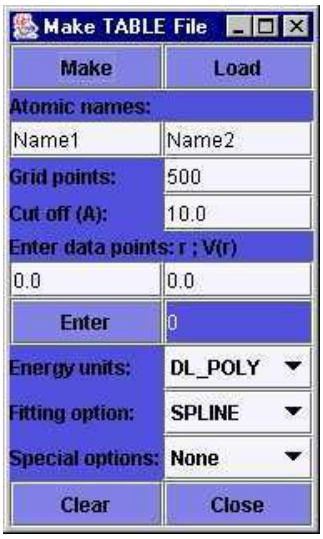

Figure 13: The Make TABLE File Panel

The prime purpose of this panel is to fit a set of data points describing a potential function, and from the fit construct a full TABLE file for DL_POLY input. Two kinds of fit are available: fitting by splines, or by gaussian functions. The selection is made from the 'Fitting Option' menu on the panel. The user must also specify, in the text boxes provided, the names of the atoms for which the pair potential is being defined. Also required are the range of the potential cut off and the number of points for the tabulation (not to be confused with the number of data points). The user may then enter the data to be fitted either by hand or by loading a data file. The **Load** button enables loading from an XY file, which will will then be fitted and a TABLE file created with the name TABSPL.n (spline) or TABGSS.n (gaussian), with n some integer. Specifying the data by hand requires the user to enter the r and V(r) coordinates, one pair at a time, into the etxt boxes provided. After each of which the **Enter** button is pressed. This can be tedious and thus error prone, and a **Clear** button is available to restart the process. When all data coordinates have been entered, clicking the **Make** button will produce the TABLE file.

The panel also has some built-in potentials under the 'Special options' menu. One is for silica (SiO_2) [60] and the other for silver iodide [77]. (These have proved useful for student exercises.) The TABLE files produced are named TABSiO2.n and TABAgI.n respectively.

The **Close** button deletes the 'Make Table' panel.

8.8.4 Display

This menu item enables the user to load configuration files with a format different from the standard DL_POLY CONFIG file. The options on its submenu are as follow.

- **CFG** - displays DL_POLY CONFIG files. This option is the same as the **New** button on the GUI button panel.
- **XYZ** - displays a standard XYZ file.
- **SEQ** - displays a protein sequence PDB file.
- **MSI** - displays a CERIUS 2 msi file.

Each of these options (except **CFG**) will also produce a CONFIG file suitable for input to DL_POLY. These appear with names CFGXYZ.n, CFGSEQ.n or CFGMSI.n respectively, with n some integer.

8.8.5 Tools

The GUI provides some utilities that are useful when building DL_POLY input files. The following are available.

1. **N_fold**
   This panel (figure 14) is useful for scaling up systems to multiples of the original, for example taking a single polymer chain and generating a layer.

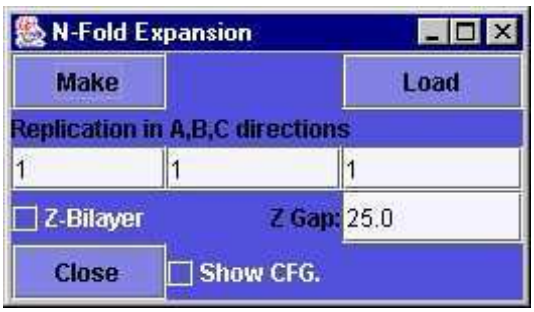

Figure 14: The N_fold Panel

   The panel has three text boxes to enter the integer scaling factors in the x, y and z direction. (For a layer, expand only in x and y.) The button **Make** can be used for a CONFIG file that is loaded already. The **Load** button will load a different CONFIG file from a browser. Both options produce a file called CFGBIG.n, where n is the suffix of the loaded CONFIG file. A check box enables the display of the expanded structure. The 'Z-bilayer' check box is used to create bilayers, by doubling the system in the z-direction. The text box 'Z gap' specifies the separation between the layers. (Note that a bilayer can also be created using the bilayer options in the 'Make Chain' application above and expanding the system using N_fold, but ignoring the bilayer option here.) The **Close** button deletes the 'N_fold' panel.

2. **Bondlengths**
   The CONFIG file builders for the GUI, such as the polymer and chain options assume certain bond lengths for particular types of bonds. This panel (figure 15) gives the user the opportunity to alter the default values.

A generic bond length is reset by typing in the new value into the appropriate text box and clicking the **Set** button. All subsequent CONFIG files will use the new bond length. Bondlengths are specified in Å.

The **Close** button deletes the 'Set Bondlengths' panel.

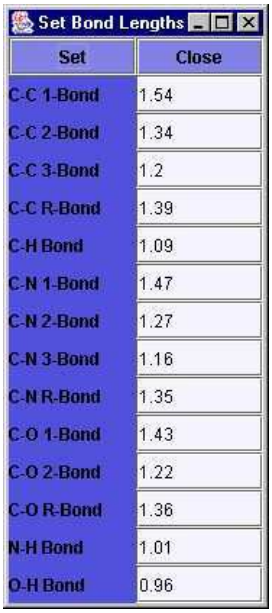

Figure 15: The Set Bondlengths Panel

3. **Add water**

The 'Add Water' panel (figure 16) inserts water into a CONFIG file to make. The water configuration is taken from a file of water molecules equilibrated at 300 K and inserted into the CONFIG file, with replication and truncation if necessary. Overlap of the water with the solute is prevented by removal of the offending water molecules.

The **Make** button will add water to an already loaded CONFIG file and the **Load** button to a file selected from a browser. The file created has the name CFGH2O.n, where n is the integer suffix of the input file. The user may specify the closest approach of water molecules to the solute and between the water molecules at the periodic boundary (necessary if the water source file has been truncated) using the labelled text boxes.

The 'Add Water' panel also has a check box to enable insertion of water in a 'slab', for example in a bilayer gap. To do this the user must specify a direction vector in the text boxes provided, and the upper and lower bounds of slab perpendicular to this vector. It is assumed the vector is drawn from the centre of the simulation cell. For example a direction vector (0,0,1) and upper and lower bounds of 3.0 and -3.0 respectively will create a water layer in the x, y direction, 6 Åwide, centred on the middle of the simulation cell.

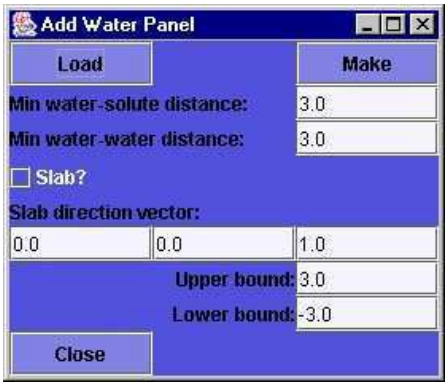

Figure 16: The Add Water Panel

The **Close** button deletes the 'Add Water' panel.

Note that in on order to display the full CFGH2O file, the Graphical window button **H2O** must be used to toggle the visibility of the water moelcules.

8.9 Execute Menu

The Execute Menu offers facilities for running the DL_POLY executable and for managing the input and output files. The menu has two items:

1. **Run DL_POLY**
2. **Store/Fetch Data**

The operation of each of these is given below.

8.9.1 Run DL_POLY

This menu item opens the 'Run DL_POLY' panel (figure 17), which hosts several buttons to control a DL_POLY execution.

The upper middle of the panel is dominated by a group of four buttons: **CONTROL**, **CONFIG**, **FIELD** and **TABLE**. Each of these buttons opens a file browser, for selection of the corresponding input file for a DL_POLY run. It is assumed that all of these files are present in the *execute* subdirectory and are the products of the GUI. Thus all potential CONTROL files will begin with the letters CNTROL, all potential FIELD files with FLD, all potential CONFIG files with CFG and all potential TABLE files (if any) with TAB. On unix systems, the file browser will list files on this basis, but Windows file browsers (to date) are not discriminating. It is the responsibility of the user to select a consistent set of input files. The selected file is copied into one of CONTROL, CONFIG, FIELD or TABLE, which are the files DL_POLY expects to be in the *execute* subdirectory at run time.

When the files have been selected, and provided the *execute* subdirectory properly contains the DLPOLY.X executable, clicking the **Run** button will start the DL_POLY program running in the background. Note that this represents a spawned process, and the program should keep running even if the GUI is closed down. There are advantages to keeping the GUI running however.

Figure 17: The Run DL_POLY Panel

The status of the job can be obtained by clicking the panel **Status** button. This will result either in a display of the job elapsed time, if still running, or a statement that the job has finished. This message will appear in the Monitor window.

The GUI will not allow two jobs to be run at the same time, since there is danger of file corruption. So all jobs must be formally killed, using the **Kill** button, after completion to permit a following run. A job may also be terminated prematurely by clicking the **Kill** button.

Neither the **Status** nor the **Kill** buttons will operate if the GUI has been closed down meanwhile.

Input and output files for DL_POLY left lying around in the *execute* subdirectory after a run may be cleared away using the **Clear** button. Files required for a subsequent run may be set up using the **Update** button, which will result in backup copies of the CONFIG and REVOLD files being taken (and given the suffix .BAK), and the files REVCON and REVIVE being renamed CONFIG and REVOLD respectively. Note that it is up to the user to amend the CONTROL file if necessary, using the CONTROL file editor under the **FileMaker** menu.

Note that because both **Clear** and **Update** may result in data loss if inappropriately used, so a dialog box will appear when either is clicked, to provide opportunity to abort the operation.

The **Close** button deletes the 'Run DL_POLY' panel.

8.9.2 Store/Fetch Data

Selection of this menu item invokes the 'Data Archive' panel (figure 18), which provides facilities for storing and retrieving DL_POLY I/O files.

The first option available on the 'Data Archive' panel is the selection of the standard test cases of DL_POLY. The user must first choose the required test case from the menu box on the panel, then clicking the **Select** button will result in the data files for this test case being copied into the *execute* subdirectory. These files may be used immediately for a DL_POLY run, since they already have the correct file names and do not need to be selected again. Note that the GUI selects only the leapfrog versions of th test data.

The user may define a storage directory under the DL_POLY *data* subdirectory and use the Data Archive panel to store DL_POLY I/O files. The user should enter a *new* directory name in the text box beside the **Store** button and then click **Store**. (This action also deletes the original files from the *execute* subdirectory.) If the nominated directory already exists, an error message appears in the Monitor window and no action is taken.

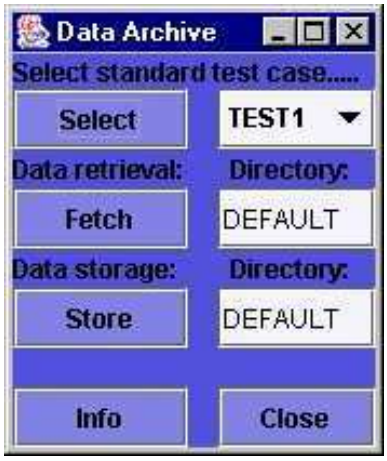

Figure 18: The Data Archive Panel

In a similar fashion the **Fetch** button will copy DL_POLY I/O files from the directory nominated in the adjacent text box into the *execute* subdirectory. The nominated directory must be in the DL_POLY *data* subdirectory.

Note that both the **Select** and **Fetch** buttons will first produce a dialog box, to alert the user to a possible data overwrite.

The **Close** button deletes the 'Data Archive' panel.

8.10 Analysis Menu

The Analysis menu provides access to a range of tools for analysing DL_POLY output. The menu items that appear are as follows:

1. **Statistics** - calculate and plot simulation data;
2. **Structure** - spatial correlation functions;
3. **Dynamics** - time correlation functions;
4. **van Hove** - space-time correlation and dynamic structure;
5. **Display** - visualisation;
6. **Tools** - utilities.

Each of these items presents a submenu of applications, which are described below.

8.10.1 Statistics

The 'Statistics' Panel invoked by this option (figure 19) allows the user to calculate average values and statistical errors of particular system variables, using information stored ion the DL_POLY STATIS file.

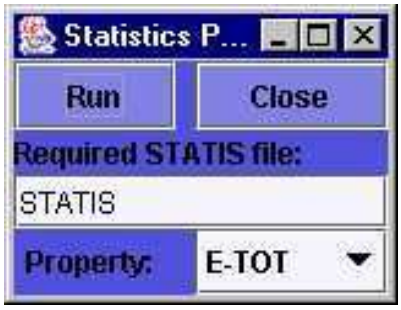

Figure 19: The Statistics Panel

The panel supports a text box for the name of the STATIS file and a menu box for the variables that may be analysed. These are:

1. E_TOT - system total energy;
2. TEMP - system temperature;
3. E_CFG - configuration energy;
4. E_VDW - van der Waals energy;
5. E_COUL - Coulombic energy;
6. VOLUME - system volume;
7. PRESS - system pressure;
8. PMF - potential of mean force virial;

The statistical calculations are initiated by the **Run** button. While calculating the average value of the selected variable, the GUI also performs a blocking analysis to obtain the optimal statistical error and the error uncertainty. The results are displayed in the Monitor window. A graph plot of the variable is produced on-screen by the GUI Graph Plotter.

8.10.2 Structure

This menu item invokes a submenu of facilities for analysing the static structure of a system. The following facilities are provided.

1. **RDF_Plot**
   The 'RDF Plotter' panel (figure 20) enables the user to plot a radial distribution function produced by DL_POLY.

   The RDF data are stored in the file RDFDAT, but if the user has renamed this file, the name in the text box available for this purpose must be changed. To produce a RDF plot all the user need do is nominate the two required atom names in the text boxes provided and click the **Plot** button. (The names are those used to label atoms in the simulation CONFIG or FIELD files.) The GUI will produce a screen plot of the RDF and also create an associated plot file named RDFn.XY, where n is some integer. The on-screen plot is produced by the GUI Graph Plotter (see section 8.12).

Figure 20: The RDF Plot Panel

Note the names of the atoms present in any CONFIG file may easily be obtained with the 'What Atoms?' facility under the Analysis/Tools menu (see section 8.10.6).

The **Close** button deletes the 'RDF Plot' panel.

2. **RDF_Calc**
The data stored in the RDFDAT file does not necessarily provide a complete account of pair correlations in a system. For example bonded pairs are not described, nor are pairs for which the interaction potential is defined as zero. The 'RDF_Calc' panel (Figure 21) provides the means to calculate these missing correlations using a DL_POLY HISTORY file. The panel can also calculate a total RDF for the system, combining all atom types.

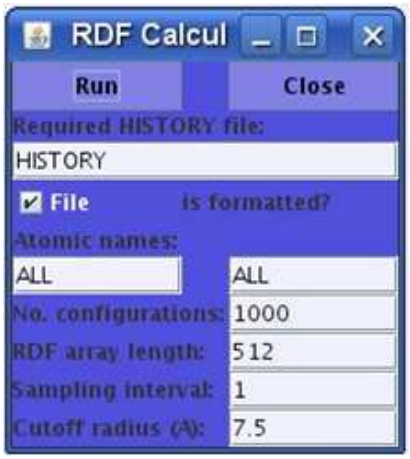

Figure 21: The RDF Calculator Panel

The data required by this panel are as follows. Firstly the name of the HISTORY file is required. (This must be a file in the *execute* subdirectory.) The default file name is HISTORY. If the file is formatted, the check box on the panel must be set. Next the user must supply the atom names for the pair correlation, as for the RDF plotter. Note that the name ALL may be used if a total RDF is required. The user must specify the required number of configurations in the HISTORY file using the associated text box. (This may exceed the actual number without harm.) Also required are the length of the RDF array (how many data points in the plot), the HISTORY file sampling interval (e.g. setting 1 will sample all configurations, 2 will

sample every other configuration, and so on) and the cut off radius (in Å) for the RDF. Text boxes are available for all of these.

Clicking the **Run** button will start the RDF program running. When finished, the program produces a file named RDFDAT.n, where n is an integer. This file may be plotted using the RDF plotter described above (section 1).

The **Close** button deletes the 'RDF Calculator' panel.

3. **S(k)**
The S(k) plotter panel (figure 22) is used to plot a structure factor, based on the RDF data in the RDFDAT file.

Figure 22: The S(k) Plotter Panel

This panel works in exactly the same way as the RDF plotter above (section 1). The only difference is that the RDF data are Fourier transformed immediately to give the structure factor. An on-screen plot of this appears, drawn by the GUI Graph Plotter (section 8.12, and a plot file SOKn.XY, with n some integer, is produced.

4. **Z_Density**
The 'Z-Density Plotter' panel (figure 23) plots the particle density of a system along the z-direction, taking data from a DL_POLY ZDNDAT file. This has particular application to layered systems, where, by convention, the layers lie in the x, y plane.

Figure 23: The Z-Density Plotter Panel

To operate this panel the user must nominate the ZDNDAT file in the appropriate text box and the name of the atom of interest. The **Plot** button produces an on-screen plot using the

GUI Graph Plotter facility (section 8.12) and also writes the plot file ZDENα.XY, where α is an integer.

Note the names of the atoms present in any CONFIG file may easily be obtained with the 'What Atoms' facility under the Analysis/Tools menu (see section 8.10.6).

The **Close** button deletes the panel.

5. **Slice**
The 'Slice' panel (figure 24) enables a user to cut a slice, or slab of atoms from REVCON file or a loaded configuration. This can be useful to isolate areas of interest for closer inspection.

To use the slice option the user must define a slice direction vector, (which is perpendicular to the slice faces,) and the upper and lower bounds of the slice measured along the chosen direction. It is assumed the direction vector starts at the centre of the REVCON file. Labelled text boxed are provided for these inputs. To perform the slice operation on a loaded configuration the **Make** button must be clicked. Alternatively the **Load** button may be used, to enable selection of a REVCON file from a browser. Both bottons produce a file CFGSLC.n, with n an integer (absent in some cases) taken from the configuration file suffix. The sliced configuration appears in the Graphics window.

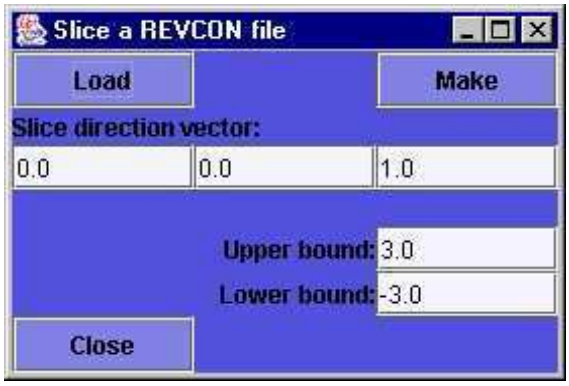

Figure 24: The Slice Panel

Note: Being an analysis tool, this facility is targetted at REVCON files, though it will work equally well on CONFIG files also.

The **Close** button deletes the 'Slice' panel.

8.10.3 Dynamics

The Dynamics submenu offers some standard time correlation functions, namely the mean square displacement (MSD), velocity autocorrelation (VAF) and force autocorrelation (FAF). These are calculated from the data in a DL_POLY HISTORY file.

1. **MSD**
The MSD Panel (figure 25) enables a multiple origin MSD calculation to be performed and the result plotted on-screen.

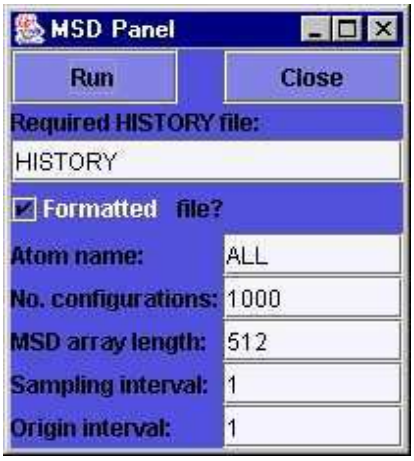

Figure 25: The MSD Panel

The user must specify the HISTORY file name in the labelled text box. The file must be in the DL_POLY *execute* subdirectory. The default name is HISTORY. A check box is used to indicate that the file is formatted or otherwise. A text box is available for the name of the atom of interest, which may be specified as ALL if a non-discriminating MSD is required. Text boxes are also provided for the required number of configurations in the HISTORY file, the MSD array length, sampling interval and origin interval. The sampling interval defines the interval between selected configurations in the HISTORY file, for example an assignment of 1 means every configuration is used, while 2 would use every second configuration and 3 every third and so on. The origin interval specifies which of the sampled configurations is to be used as an origin for a MSD array. Thus a specification 1 means every sampled configuration is used as an origin, 2 means every second sampled configuration is used etc. By a quirke of bookkeeping the MSD array length must be divisible by the origin interval. The GUI will enforce this if necessary.

Clicking the **Run** button starts the MSD calculation. On completion the MSD is plotted by the GUI Graph Plotter (see section 8.12) and a plot file MSDn.XY created, with n an integer.

The **Close** button deletes the MSD panel.

2. **VAF**
   The VAF panel is identical to the MSD panel in appearance and operation. Its outputs are a plot of the Velocity Autocorrelation Function data and a plot file: VAFn.XY, with n an integer. Please consult the above section describing the MSD panel. Note that the HISTORY file must contain velocity data, and DL_POLY should be directed to produce this when the simulation is performed.

3. **FAF**
   The FAF panel is identical to the MSD panel in appearance and operation. Its outputs are a plot of the Force Autocorrelation Function data and a plot file: FAFn.XY, with n an integer. Please consult the above section describing the MSD panel. The HISTORY file must, of course, contain force data.

8.10.4 van Hove

This menu item provides a selection of density correlation tools of the kind pioneered by van Hove. (i.e. correlations in both space and time). The facilities available are:

1. **Gs(r,t)**

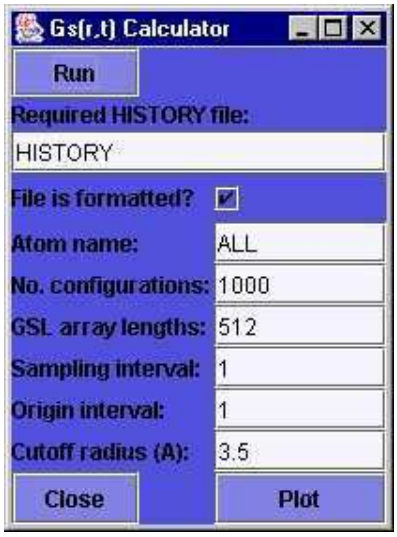

Figure 26: The Gs(r,t) Calculator Panel

The panel for the van Hove self correlation function (see figure 26) resembles that for the MSD above, but requires an additional control parameter: the cutoff radius, which is the distance over which the spatial correlations are to be evaluated. The calculations are commenced when the RUN button on the file browser is clicked and may take several minutes to complete. The results are stored in a file: HOVGSL.n, where n is an integer. Unlike the MSD, VAF and FAF panels this panel does not produce a graph window automatically, since, potentially, a large number of Gs(r,t) functions are produced. (The actual number is announced in the Monitor Window when the calculation finishes.) It is necessary to use the **Plot** button to invoke the van Hove Plotter and select which of the many functions is required for plotting by entering the sequence number in the text box (see figure 27). The selected plot appears on-screen in the Graph Plotter and each plot produces a plot file: HOVm.XY, where m is the sequence number of the van Hove function in the parent HOVGSL.n file.

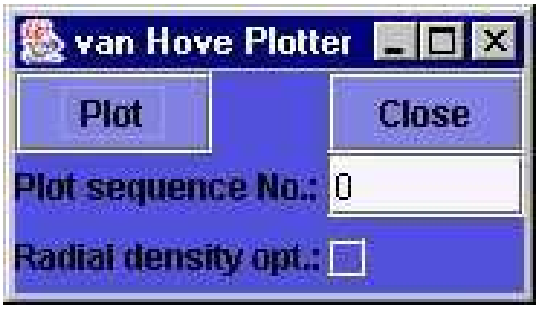

Figure 27: The van Hove Plotter Panel

2. **Gd(r,t)**
The van Hove distinct correlation function panel (figure 28) resembles that for the self correlation function, except that an additional atom name is required, since this is a pair correlation function. The name ALL may be used for indescriminate correlation functions. The calculations are performed *in a background job*, which may be monitored using the **Status** button or terminated with the **Kill** button on the panel. When the job is finished, plotting the distinct correlation functions, (which are stored in the file HOVGDF.n,) is identical to the self correlation function case.

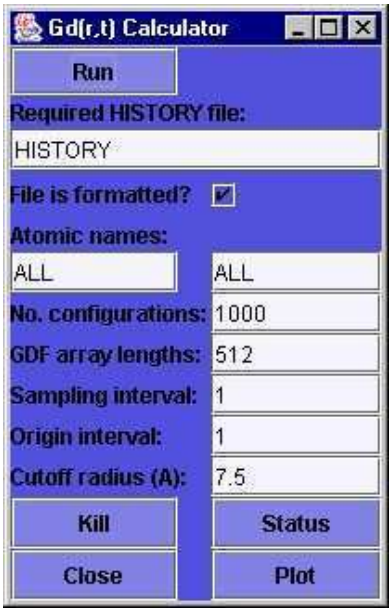

Figure 28: The Gd(r,t) Calculator Panel

3. **S(k,w)**
The dynamic structure factor panel (figure 29) also operates in a manner resembling the self correlation function panel, however there are important differences. Firstly, the calculation (*which runs in the background*) will accept only *formatted* HISTORY files. It does not require the names of the atoms, but does require the user to specify whether or not the atomic charges are to be used (to calculate charge density as opposed to particle density). A check box is available for this purpose. The maximum k vector is specified by an integer index, which determines the maximum in all three principal directions as in:

$$\underline{k} = 2\pi/L(\ell, m, n)^{\dagger},$$

where ℓ, m, n are the integers concerned and L is the cell width. The calculation proceeds via the particle density $\rho(k,t)$, through the intermediate scattering function $F(k,t)$ to the dynamic structure factor $S(k,\omega)$. These stages respectively produce files called SPCDEN, DENFKT and DENSKW. The latter two may be displayed by clicking the **Plot** button, which invokes the dynamic structure factor plotting panel. The panel supports **Status** and **Kill** buttons to help manage the background job.

Figure 29: The S(k,w) Calculator Panel

The plotting panel (figure 30) resembles the van Hove plotter, but the choice of function to be plotted is specified by the three k-vector indices, for which individual text boxes are provided.

Figure 30: The S(k,w) Plotter Panel

8.10.5 Display

The Display menu item offers a facility for displaying REVCON files and (re-)plotting various graphs.

1. **CONFIG**
   This option displays the contents of the CONFIG file used in a DL_POLY simulation. The selection invokes a display the CONFIG file.

2. **REVCON**
   This option displays the contents of the REVCON file produced at the end of a DL_POLY simulation. Its function is entirely analogous to the Display CFG option in the **FileMaker** menu above, and the **New** button on the Graphics Window. The action invokes a file browser for selection of the REVCON file, and can also be used to display CONFIG files.

8.10.6 Tools

1. **What Atoms?**
   The 'What Atoms?' menu item provides a mechanism by which the user may conveniently determine the different types of atom that occur in a CONFIG file. It invokes a file browser with which the required CONFIG file may be selected. A list of atom types within the selected file then appears in the Monitor window. This facility is useful for analysis purposes, as the atom types (names) are frequently required by the GUI analysis tools.

2. **Plot**
   This option invokes a simple file browser, which allows the user to select any plot file (i.e. with a name ending in .XY e.g. abc.XY) to display with the Graph Plotter.

8.11 Information Menu

The Information menu provides some on-line information of a semi-useful nature. The information is displayed in the Monitor window.

1. **About DL_POLY**
   Note on authorship and ownership of DL_POLY.

2. **Disclaimer**
   Software writer's incantation to ward off litigation.

3. **Licence**
   View of the DL_POLY licence on-screen.

4. **Acknowledgements**
   Acknowledgements and thanks to various people and organisations.

5. **MINIDREI**
   The contents of the MINIDREI Dreiding data file.

6. **MINIOPLS**
   The contents of the MINIOPLS OPLS data file.

7. **CERAMICS**
   The contents of the CERAMICS ceramic data file.

8. **Clear Text**
   Clear the Monitor window.

8.12 The GUI Graph Plotter Window

The Graph Plotter window (figure 31) is invoked automatically by some of the Analysis tools described above and also by selecting the Analysis/Display/Plot menu item. The scale of the graph is calculated automatically and the window provides facilities to display or print a graph plot and also perform some editing.

The Graph Plotter presents a large drawing area, with three text boxes at its base and a number of buttons stacked vertically on the right hand side. Most of the functionality of the plotter is controlled by the buttons, which are as follows:

- **Load** - load and plot a new XY file;

- **Spline** - Fit data points with spline functions and plot result;
- **Dots** - Show/hide dots marking data points;
- **Lines** - Set plot line thickness.
- **Print** - Open a print dialog box for printing;
- **Zoom** - Zoom in on selected region of plot (marked with drag box);
- **Clear** - Delete plot and clear arrays;
- **Close** - Delete Graph Plotter window.

Figure 31: The GUI Graph Plotter

In addition to the button operations, the text boxes at the foot of the drawing area allow the user to change the graph annotation. From the left, the first box defines the x axis notation, the second box the y-axis notation and the last defines the graph title. The text in any of these may be edited. Hitting $< Return >$ will insert the changed text into the plot.

The GUI Graph Plotter reads and writes data files with the following format:

1. **Record 1: File header record**
   Record starts with the '#' character followed by user text;

2. **Record 2: Plot title record**
   Record starts with the '#' character followed by text defining the plot title;
3. **Record 3: X-axis label**
   Record starts with the '#' character followed by text defining x-axis label;
4. **Record 4: Y-axis label**
   Record starts with the '#' character followed by text defining y-axis label;
5. **Records 5+: Data points**
   Records defining the x and y data points of the plot. x and y must be real numbers separated by at least one space. Scientific (E) number format is acceptable.
6. **Last record: terminator**
   The data must be terminated with a '&' character.

This format is equivalent to the common XY format used by most data processing packages.

8.13 The Molecular Editor

8.13.1 Introduction

The Molecular Editor gives the user the ability to construct complex organic molecules and replicate them to build a system that can be simulated by DL_POLY. The Editor is invoked by clicking the **Edt** button on the right of the GUI, when in View mode. The appearance of the GUI is shown in figure 32. (Similarly, the Editor may be closed by clicking the **Edt** button in Edit mode.) Invoking the Editor has the following consequences:

- An extra column of buttons appears on the right of the GUI and the way the buttons work is altered. In general buttons no longer produce a direct action, they merely activate an edit mode. The action is usually performed by mouse and cursor operations in the Graphics Window.
- An extra menu: 'Editor', appears on the menu bar. This is used to set the editor defaults.
- The rendering of structures in the Graphics Window changes - atoms are drawn smaller.
- A small crosswire appears in the centre of the Graphics Window. This is the 'system centre' - the centre of the MD cell and the point about which molecules are rotated when required.

The significance of these features will become apparent in the following text.

8.13.2 The Buttons of the Molecular Editor

8.13.2.1 The Mode of Action of the Buttons

A common feature of the buttons in the Edit mode is that clicking on them puts the GUI into a sub-edit mode, which is identified by a text banner appearing in the top left of the Graphics Window. While the GUI is in a subedit mode the molecular structure in the Graphics Window may be edited in the selected manner. Clicking the same button, switches off the sub-edit mode. This is sometimes accomplished by clicking another button, but this action may sometimes activate the second option to work concurrently with the first. The default sub-edit mode is NULL. To obtain the required editing operation, the mouse cursor should be moved to the Graphics Window, where

the options of clicking or dragging the cursor may be employed in accordance with the selected edit operation. The following properties should be noted.

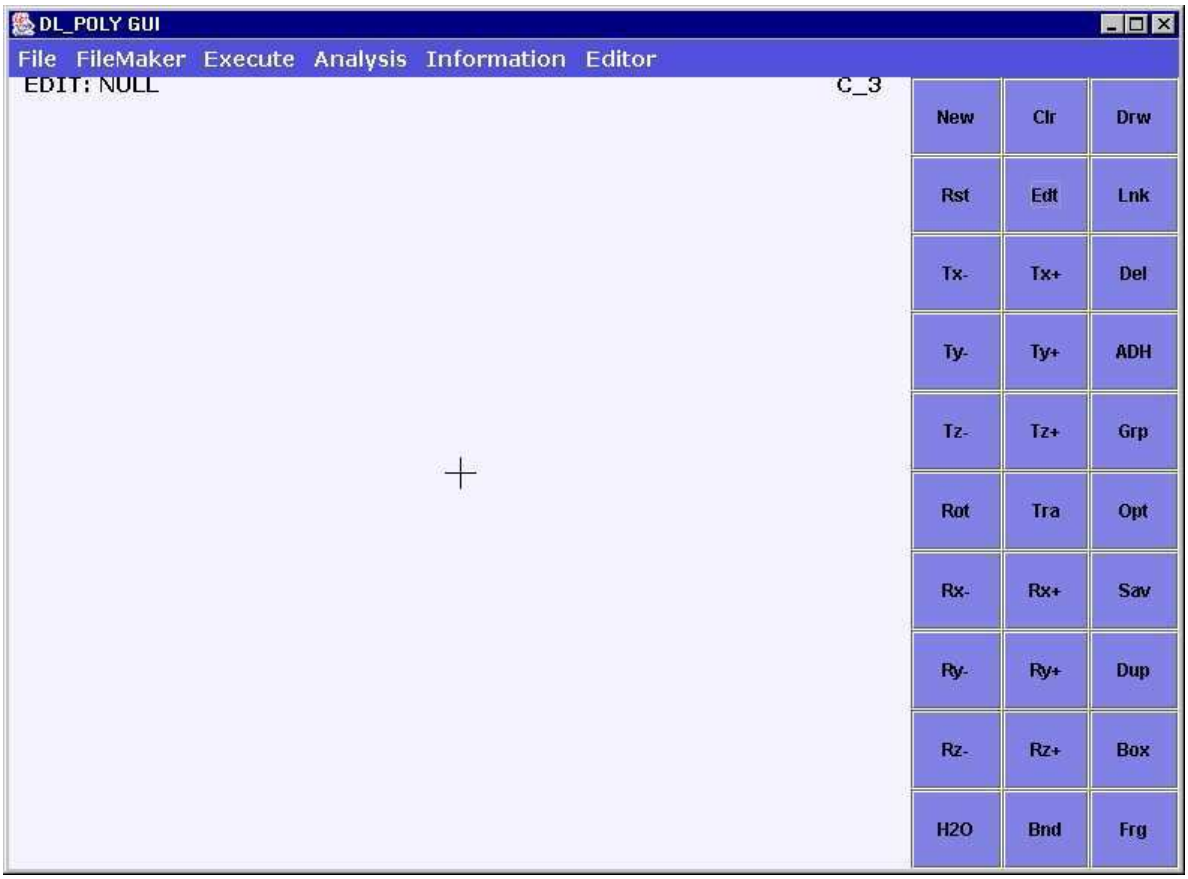

Figure 32: The GUI with the Molecular Editor activated.

- For operations that respond to a mouse click, clicking the mouse in an empty part of the Graphics Window will normally perform the operation on the whole structure, or if an atom group has been created (see below), it will operate on that group. If the user clicks on an individual atom, the operation will be applied to that atom only.
- Similarly, for operations that respond to a mouse drag, starting the drag in an empty part of the Graphics Window will perform the operation on the whole structure, or on an atom group if one has been created. If the user starts the drag on an individual atom, the operation will be applied to that atom only.

In the following sections the new buttons that appear in the Edit mode are described separately from those originally present (though with modified function) in the View mode. These sections identify the function of the buttons only. How they are used is described in section (8.13.4).

8.13.2.2 The 'New' Buttons

The new buttons are as follows:

- **Drw** - activates the drawing mode in the editor.

- **Lnk** - draw a bond between two atoms.
- **Del** - delete an atom or group of atoms.
- **ADH** - add or delete hydrogen atoms.
- **Grp** - create a group of atoms for editing purposes.
- **Opt** - optimise the structure.
- **Sav** - Save the structure in a file.
- **Dup** - Duplicate a group of atoms.
- **Box** - Draw a MD cell around structure.
- **Frg** - Insert a predefined molecular fragment.

8.13.2.3 The Modified 'Old' Buttons

The buttons retained from the View mode have broadly the same meaning as before, though the actual function may be different. The amended functions are as follows:

- **New** - allows user to read a CONFIG file, in this case for editing.
- **Cls** - deletes the structure being edited and clears the image from the screen.
- **Rst** - restores the last structure saved by the editor.
- **Edt** - activates/deactivates the Edit mode.
- **Tx-** - moves (translates) the structure to the left.
- **Tx+** - moves the structure to the right.
- **Ty-** - moves the structure towards the observer.
- **Ty+** - moves the structure away from the observer.
- **Tz-** - moves structure down.
- **Tz+** - moves structure up.
- **Rot** - rotates the structure to follow the dragged cursor.
- **Tra** - moves the structure to follow the dragged cursor.
- **Rx-** - rotates the structure clockwise about the x axis.
- **Rx+** - rotates the structure anticlockwise about the x axis.
- **Ry-** - rotates the structure anticlockwise about the y axis.
- **Ry+** - rotates the structure clockwise about the y axis.
- **Rz-** - rotates the structure clockwise about the z axis.
- **Rz+** - rotates the structure anticlockwise about the z axis.

- **H2O** - unchanged - toggles the visibility of water molecules.

In the above it should be noted that the sub-edit modes that the translation and rotation buttons act on the molecular structure and not just its image. Real change results from these operations, unlike what happens in View mode. Note that the **Ty+** and **Ty-** buttons, do not 'zoom' out and in as happens in the View mode. In Edit mode the structure is displaced only about 0.1 A in either direction and is used to help with optimisation, as will be shown later. Note also that it is permissible to click the axis rotation buttons (**Rx+**, **Ry-** etc) after the **Rot** button to obtain a mouse controlled rotation about the indicated axis. Normally the axis rotation buttons rotate the structure by a prescribed fixed amount. Finally and importantly, please note there is no UNDO button, so you are recommended to save your edits frequently!

8.13.3 The Editor Menu

The Editor menu appears only in the Edit mode and provides the user with a means for setting the Edit mode defaults. The menu items available are:

- **Atoms** - sets the default atom for the Draw sub-edit mode;
- **Box** - sets the default MD cell for the Box sub-edit mode;
- **Fragment** - selects the molecular fragment for the Fragment sub-edit mode.

These options are described below.

1. **Atoms**
   This item provides a sub-menu of possible atom types that may be used in drawing molecular structures. The current list includes: H_, C_3, C_2, C_1, C_R, O_3, O_2, N_3, N_2, N_1, P_3, P_2, S_3, S_2, which are atom types consistent with the Dreiding force field. Selection of one of these will make it the default atom for drawing molecular structures. The initial default is C_3.

2. **Box**
   This item provides a sub-menu of types of MD cell to contain the edited structure. The following are available: Cubic, orthorhombic, truncated octahedral, rhombic dodecahedral and hexagonal. Selection of one of these will define the shape of the MD cell the GUI will use when the Box sub-edit mode is activated.

3. **Fragment**
   This item provides a sub-menu of molecular fragments for insertion into the structure using the Fragment sub-edit mode. The fragments currently available are: Search (default), alanine, benzene, glucose, i-butane, naphthalene, styrene, c-hexane, n-butane, n-hexane, n-decane. Selection of one of these will make it the default molecular fragment to insert into the structure during the Fragment sub-edit mode. The one exception is the Search option, which will open a file browser for selection of an appropriate CONFIG file for insertion.

8.13.4 The Sub-Edit Modes

1. **The Null Sub-edit Mode**
   This mode is the default, to which the GUI returns when any current edit sub-mode is deactivated. It has the following properties:

- If the user clicks on an atom, the atom will be highlighted by a red halo. Also the symbol and number of the atom selected will be printed in the Monitor Window.
- Clicking on a second atom will both highlight the atom, and print its symbol and number in the Monitor Window, together with its distance from the first atom (i.e. bond length determination).
- Clicking on a third atom will both highlight the atom, and print its symbol and number in the Monitor Window, together with its distance from the second atom and the angle between the line linking the first and second atoms and that linking the second and third atoms (i.e. bond angle determination at second atom).
- Clicking any further atom will cause highlighting of the atom and printing of the distance to the previous atom and the angle at that atom in the Monitor Window. Any number of atoms may be clicked in succession, but only three atoms will ever be highlighted.
- Clicking an empty part of the Graphics Window, will de-highlight all selected atoms.
- A double click on any atom will result in its substitution by whatever atom type is the current default.

2. **The Draw Sub-edit Mode**
   Clicking the **Drw** button activates the Draw sub-edit mode. In this mode clicking in the Graphics Window will result in the insertion of an atom of the default type (as defined using the Editor menu). Clicking in another location will create a second atom formally linked to the first via a bond, which is also drawn. Subsequent clicks add additional atoms, each linked to the previous atom. This linking between atoms can be stopped by clicking on an existing atom. The next atom added will not be linked to the previous one, though subsequent additions will continue the linking. Bonds may be edited using the Link sub-edit mode (see below). Clicking the **Drw** button while in the Drawing sub-edit mode, will deactivate the drawing. Note that an isolated (unlinked) atom is drawn in the X-Z plane with zero Y coordinate. If it is linked to a preceding atom however, it will take the Y coordinate of that atom.

3. **The Link Sub-edit Mode**
   Clicking the **Lnk** button activates the Link sub-edit mode. In this mode, the user may click on any two atoms and a link (bond) will be drawn between them. If a link already exists, it will be deleted. In both operations, the first atom clicked is highlighted. In this way links between atoms may be added and removed. Clicking the **Lnk** button while in this mode will deactivate the Link mode.

4. **The Delete Sub-edit Mode**
   This mode is activated by the **Del** button. In this mode the following operations are possible.

   - If an atom is clicked, it is deleted from the structure and any links to that atom from other atoms in the structure are also deleted.
   - If an atom group is defined, clicking anywhere in the Graphics Window will cause the deletion of the entire group and all links between its constituent atoms and the surviving atoms. Clicking the **Del** button while in this mode will deactivate the Delete mode.

5. **The Add/Delete Hydrogen Sub-edit Mode**
   The Add/Delete Hydrogen sub-edit mode is activated by the **ADH** button. Its mode of operation is as follows:

- Clicking on a single atom will add hydrogen atoms and associated bonds to the atom, provided that it has none already and its formal valency is unsatisfied.
- Clicking an empty part of the Graphics Window will add hydrogen atoms and associated bonds to all atoms, provided that no atom is linked to any hydrogen atoms already and their formal valencies are unsatisfied. If any atom is already linked to one or more hydrogen atoms, all hydrogen atoms will be deleted (in which case a second click will restore all required hydrogen atoms to satisfy the valency requirements).
- Clicking an empty part of the Graphics Window when an atom group has been defined will add or delete hydrogens as in the previous case, but confining the operation to the atom group only.
- Clicking the **ADH** button while in this mode will deactivate the Add/Delete Hydrogen mode.

6. **The Group Sub-edit Mode**
   This mode is activated by clicking the **Grp** button. In this mode the user may isolate a group of atoms for special treatment as follows:

   - Clicking on any (unhighlighted) atom will add that atom to the group.
   - Clicking on an empty part of the Graphical Window and dragging the mouse will draw a square in the window, within which all atoms are to be included in the group. The release of the mouse button will cause all atoms in the drawn square to be highlighted.
   - Clicking any highlighted atom will cancel the group.
   - Clicking the **Grp** button while in this mode will deactivate the mode, leaving the grouped atoms highlighted. This group may then be edited independently of the remaining atoms in the system. To cancel the grouping, it is necessary to enter the Group sub-edit mode again and click one of the highlighted atoms. Click the **Grp** button once again to restore the Null mode.

7. **The Optimise Sub-edit Mode**
   The Optimise sub-edit mode is entered by clicking the **Opt** button. Thereafter the structure in the Graphics Window may be optimised by:

   - Clicking an empty part of the Graphics Window (provided no atom group has been created) will cause the optimisation process to commence. The whole structure is optimised with respect to bond lengths and bond angles (not including dihedral angles). It may be necessary to click the window several times to optimise a complicated structure.
   - If an atom group has been created, that group alone will be optimised if the window is clicked. If the group is itself connected to other atoms, these connections will be optimised with regard to bond length only. Caution: when using the optimisation, the user should be aware that convergence of the process does not necessarily mean the global minimum has been found. In particular 2 dimensional ring structures may become more stable if one or more atoms are displaced out of the plane before optimisation is undertaken. The **Tx**, **Ty** and **Tz** buttons are useful for this purpose.
   - Hint: It is sometimes advantageous to delete all hydrogen atoms in the system using the Add/Delete Hydrogen option, as this will speed up convergence. Restoring hydrogen atoms afterwards will automatically optimise their positions.
   - Clicking the **Opt** button while in this mode will deactivate the optimisation mode.

8. **The Save Button**
   Clicking the **Sav** button causes the GUI to write the current structure as a DL_POLY CONFIG file named CFGEDT.n, where n is an integer, which is sequentially increased with each save. Note the **Sav** button does not open a sub-edit mode. If the user quits the Molecular Editor without saving the structure, it is automatically saved in a backup file named CFGEDT (with no number).

9. **The Duplicate Sub-edit Mode**
   The Duplicate sub-edit mode is activated by clicking the **Dup** button. Clicking the Graphics Window in this mode will result in the following:

   - The duplication of any highlighted group, the new group becoming highlighted in the process.
   - The GUI switching to the Move sub-edit mode (normally activated by the **Tra** button), so that the newly created group may be conveniently relocated as desired.
   - Since the new group remains highlighted, the duplication operation may be repeated as many times as required, though the **Dup** button will need to be re-clicked for each case.
   - Clicking the **Dup** button while in this mode will deactivate the Duplicate mode.

10. **The Box Sub-edit Mode**
    Clicking the **Box** button will put the GUI into the Box sub-edit mode, which will insert an MD cell shape into the Graphics Window. This may be resized by dragging the mouse across the window. The orthorhombic and hexagonal options may be resized in independent, mutually perpendicular directions if required. The cell size becomes fixed when the Box sub-edit mode is deactivated. If the system already has a MD cell when this option is selected, the existing box will be replaced by whatever default shape has been defined through the Editor menu. Clicking the **Box** button while in this mode will deactivate the Box mode.

11. **Fragment Insertion**
    Before using the **Fra** button, the user must select a fragment from the Editor/Fragment menu. A set of possible structures is available, but this may be augmented by selecting the Search option, which will open a file browser allowing the user to select an existing DL_POLY CONFIG file as the required insertion. Clicking the **Fra** button will activate the Fragment Insertion sub-mode. If the user then clicks the Graphics Window the following will happen:

    - Insertion of the selected fragment into the Graphics Window.
    - The grouping and highlighting of the inserted fragment.
    - Switching to the Move sub-edit mode for relocation of the fragment.

    As with the Duplicate sub-edit mode this procedure may be repeated as often as desired.The current set of fragments available in the GUI is somewhat eclectic, but serve to show what is possible. It is not difficult to edit the GUI source to permit insertion of alternative fragments, for example -amino acids, carbohydrates, polymer monomers etc. The user is encouraged to do this if it suits his or her purpose. The Fragment Insertion sub-edit mode is deactivated by clicking the **Fra** button once again.

Chapter 9

Example Simulations

Scope of Chapter

This chapter describes the standard test cases for DL_POLY Classic, the input and output files for which are in the *data* sub-directory.

9.1 DL_POLY Examples

9.1.1 Test Cases

The following example data sets (both input and output) are stored in the subdirectory *data*. Two versions are provided for the Leapfrog (LF) and Velocity Verlet (VV) algorithms respectively, so that you may check that your version of DL_POLY is working correctly. All the jobs are short and should require no more than a few minutes execution time, even on a single processor computer. The test cases can be chosen by typing

select n a

from the *execute* directory, where n is the number of the test case and a is either LF, VV, CB or RB. The *select* macro will copy the appropriate CONTROL, CONFIG, FIELD and if necessary the TABLE or TABEAM files to the *execute* directory ready for execution. The output files OUTPUT, REVCON and STATIS may be compared with the files supplied in the *data* directory.

The example output files provided in the *data* directory were obtained on 8 processors of an Intel Xeon (Woodcrest) cluster with the following characteristics:

- Intel Xeon Dual-core processor, 3GHz (32 compute nodes, 2x2 cores each; master node; 2 NFS file servers);
- 8 GB memory per node;
- SUSE LINUX 10.1 with kernel 2.6.16.21-0.25-smp;
- Intel Compilers (version 10.1), Intel Cluster Tool kit including Intel MPI 3.0, MKL 9.1, and VTune;
- InfiniPath interconnect (software stack 2.1).

It should be noted that the potentials and the simulation conditions used in the following test cases are chosen to demonstrate functionality only. **They are not necessarily appropriate for serious simulation of the test systems.** Note also that the DL_POLY Classic Graphical User Interface [9] provides a convenient means for running and viewing these test cases.

9.1.1.1 Test Case 1: $KNaSi_2O_5$

Potassium Sodium disilicate glass ($NaKSi_2O_5$) using two and three body potentials. Some of the two body potentials are read from the TABLE file. Electrostatics are handled by a multiple timestep Ewald sum method. Cubic periodic boundaries are in use. NVE ensemble.

9.1.1.2 Test Case 2: Metal simulation with Sutton Chen potentials

FCC Aluminium using Sutton-Chen potentials. Temperature is controlled by the method of Gaussian constraints. NVT Evans ensemble.

9.1.1.3 Test Case 3: An antibiotic in water

Valinomycin in 1223 spc water molecules. The temperature is controlled by a Nosé-Hoover thermostat while electrostatics are handled by a screened reaction field Coulombic potential. The water is defined as a rigid body while bond constraints are applied to all chemical bonds in the valinomycin. Truncated octahedral boundary conditions are used. NVT Hoover ensemble.

9.1.1.4 Test Case 4: Shell model of water

256 molecules of water with a polarizable oxygen atom using adiabatic dynamics. Temperature is controlled by the Berendsen thermostat while electrostatics are handled by the reaction field method with a "charge group" cutoff scheme. "Slab" period boundary conditions are used. The water molecule (apart from the shell) is treated as a rigid body. NVT Berendsen 'ensemble'.

9.1.1.5 Test Case 5: Shell model of $MgCl_2$ at constant pressure

Adiabatic shell model simulation of $MgCl_2$. Temperature and pressure are controlled by a Berendsen thermostat and barostat. An Ewald sum is used with cubic periodic boundary conditions. NPT Berendsen 'ensemble'.

9.1.1.6 Test Case 6: PMF calculation

Potential of mean force calculation of a potassium ion in SPC water. Electrostatics are handled by the Ewald sum. The water is treated as a constrained triangle. PMF 'ensemble'

9.1.1.7 Test Case 7: Linked rigid bodies

8 biphenyl molecules in cubic boundary conditions. Each phenyl ring is treated as a rigid body, with a constraint bond to the other ring of the molecule. In the centre of each ring are three massless charge sites which imparts a quadrupole moment to the ring. NVE ensemble.

9.1.1.8 Test Case 8: An osmosis experiment with a semi permeable membrane

The membrane is a collection of tethered sites interconnected by harmonic springs. There are no electrostatic forces in the system. The simulation is run with the Hoover anisotropic constant presure algorithm. (NST Hoover ensemble.)

9.1.1.9 Test Case 9: A surfactant at the air-water interface

The system is comprised of 32 surfactant molecules (trimethylaminododecane bromide or TAB-C12) arranged either side of a slab of 342 water molecules approximately 30 $\mathring{A}$ thick. The surfactant chains are treated with rigid bonds and the water molecules are treated as rigid bodies. The TAB headgroup has fractional charges summing to +1 (the bromide ion has charge -1). The Ewald sum handles the electrostatic calculations. The short range forces are taken from the Dreiding force field. NVE ensemble.

9.1.1.10 Test Case 10: DNA strand in water

This system consists of a strand of DNA 1260 atoms in length in a solution of 706 (SPC) water molecules. The DNA is aligned in the Z-direction and *hexagonal prism* periodic boundary conditions applied. The electrostatic interactions are calculated using the Smoothed Particle Mesh Ewald method. Note that the system has a strong overall negative charge which is strongly anisotropic in distribution. The short range forces are taken from the Dreiding force field, and constraints are used for all covalent bonds. For simplicity H-bonds are treated as harmonic bonds with an equilibrium bondlength of 1.724 $\mathring{A}$. NVE ensemble.

9.1.1.11 Test Case 11: Hautman-Klein test case 1

The system consists of 100 short chain surfactant molecules in a layer simulated under NVE conditions . The total system size is 2300 atoms and the XY periodicity is a square. The Dreiding force field describes the molecular interactions. All bonds are harmonic and all atoms are explicit. The link-cell algorithm is in operation. NVE ensemble.

9.1.1.12 Test Case 12: Hautman-Klein test case 2

This is a simple test system consisting of 1024 charged particles in a layer under NVE conditions. Lennard Jones forces are used to keep the atoms apart. The similation cell is square in the XY plane. NVE ensemble.

9.1.1.13 Test Case 13: Carbon Nanotube with Tersoff potential

This system consists of 800 carbon atoms in a nanotube 41.7 A in length. The MD cell is orthorhombic and square in the XY plane. The integration algorithm is NPT Berendsen. This is a test for the Tersoff potential. NPT Berendsen 'ensemble'.

9.1.1.14 Test Case 14: Carbon Diamond with Tersoff potential

This is another test of the Tersoff potential, this time for the carbon diamond structure consisting of 512 atoms. A cubic MD cell is used with a NST Hoover integration algorithm. NST Hoover ensemble.

9.1.1.15 Test Case 15: Silicon Carbide with Tersoff potential

This is an alloy system consisting of 2744 atoms of silicon carbide in a diamond structure. The potential function used is the Tersoff potential. The integration algorithm is NPT Hoover and the initial MD cell is cubic. NPT Hoover ensemble.

9.1.1.16 Test Case 16: Magnesium Oxide with relaxed shell model

Relaxed shell model of magnesium oxide with 324 sites. The lattice is cubic and the integration algorithm is NST Berendsen. NST Berendsen 'ensemble'.

9.1.1.17 Test Case 17: Sodium ion in SPC water

A simple simulation of a sodium ion in 140 SPC water molecules (421 sites in all). The water molecules are treated as rigid bodies.The algorithm is the NVE ensemble and the Ewald sum handles the electrostatic forces. The MD box is cubic. NVE ensemble.

9.1.1.18 Test Case 18: Sodium chloride molecule in SPC water

This system resembles test case 17, except that a sodium chloride ion pair is dissolved in 139 SPC water molecules (419 sites in all). The MD cell is cubic and the water molecules are treated by constraint dynamics in the NVT Evans scheme. Ewald's method handles the electrostatics. NVT Evans ensemble.

9.1.1.19 Test Case 19: Sodium chloride molecule in SPC water

This is a repeat of test case 18, except that half of the water molecules are treated using constraint dynamics and the rest by rigid body dynamics. The integration algorithm is NPT Hoover. NPT Hoover ensemble.

9.1.1.20 Test Case 20: Linked benzene ring molecules

This test consists of pairs of benzene rings linked via a rigid (constraint) bond. Each molecule has 22 atoms and there are 81 molecules, making a total of 1782 sites. The benzene rings are treated in a variety of ways in the same system. In one third of cases the benzene rings and hydrogens form rigid groups. In another third the carbon rings are rigid but the C-H bonds are treated via constraints. In the final third, the C-H bonds are fully flexible and the rings are rigid. The MD cell is orthorhombic (nearly cubic) and the integration is NPT hoover. NPT Hoover ensemble.

9.1.1.21 Test Case 21: Aluminium metal with EAM potential

This case presents an example of the use of the EAM potential for metals, in this case aluminium. The system is 256 atoms and runs under a berendsen NPT enemble.

9.1.1.22 Test Case 22: Copper metal with EAM potential

Another example of a metal with an EAM potential. 256 copper atoms under a Berendsen NPT ensemble.

9.1.1.23 Test Case 23: Copper-Gold (3/1) alloy with Gupta potential

This is an example of the analytical Gupta potential applied to a copper-gold alloy with a 3/1 Cu/Au ratio. The system consists of 256 atoms in total running under the NVE ensemble.

9.1.1.24 Test Case 24: Iron metal with Finnis Sinclair potential

In this example the analytical Finnis-Sinclair potential is applied to iron. The system consists of 250 iron atoms and runs under a Berendsen NPT ensemble.

9.1.1.25 Test Case 25: Nickel-Aluminium (1/1) alloy with EAM potential

Another example of an alloy using the EAM potential. This is a Nickel-Aluminium alloy in the 1/1 ratio. The NVE ensemble is used and the system has 432 atoms.

9.1.1.26 Test Case 26: Nickel metal with EAM potential

Another EAM simulation of a metal. 256 Nickel atoms under the Berendsen NPT ensemble.

9.1.1.27 Test Case 27: Calcite

NVE simulation of 420 molecules (2100 atoms) of calcium carbonate in the calcite crystal structure. The carbonate anion is handled as a flexible unit with Morse potential bonds and harmonic bond angles. NVE ensemble.

9.1.1.28 Test Case 28: Optimisation of Ice VII structure

432 SPC water molecules are arranged in a thermally excited Ice VII structure and the congugate gradient method is used to optimise the structure to recover the perfect crystal form. Both rigid body (RB) and constraint bond (CB) models are used to define the water molecule structure. The optimisation proceeds to zero force convergence.

9.1.1.29 Test Case 29: Programmed minimisation of Ice VII structure

This test is a repeat of Test Case 28, except that the structural optimisation proceeds via a programmed minimisation involving alternating periods of molecular dynamics and conjugate gradient minimisation. Once again both rigid body (RB) and constraint bond (CB) models are used to define the water molecule structure and conjugate gradient optimisation proceeds to zero force convergence.

9.1.1.30 Test Case 30: Zero Kelvin structure optimisation of DNA

The DNA structure of Test Case 10 (1260 atoms) is here placed in a vacuum and a zero Kelvin optimisation is applied to reduce the overall system energy. The smoothed particle mesh method is used to handle the electrostatics.

9.1.1.31 Test Case 31: Linear molecule fluid

NPT Hoover simulation of a fluid consisting of 675 linear molecules (parameters approximate a polyacetylene chain). A 6 site rigid body is used to represent the molecules. 4050 atoms. NPT ensemble.

9.1.1.32 Test Case 32: TAD Simulation of Diffusion in Solid Argon

The TAD method is applied to Lennard Jones argon. A crystal of 255 argon atoms (FCC lattice plus one vacancy) is simulated in the NVE ensemble.

9.1.1.33 Test Case 33: BPD Simulation of Diffusion in Solid Sodium Chloride

Bias potential dynamics is applied to a crystal of sodium chloride with the rocksalt structure. NVE ensemble. 998 ions are present and two vacancies in a neutral structure. BPD is used to investigate the diffusional hops and determine the activation energies.

9.1.1.34 Test Case 34: Energy Decomposition in Liquid DMSO

The energy decomposition (or solvation energy) facility is used to provide a breakdown of the molecular configuration energy terms occuring in liquid dimethyl sulfoxide (DMSO). The basic ensemble is obtained from Berendsen's NVT algorithm. The DMSO molecule has flexible angles but rigid (constraint) bonds. 512 molecules are present in the system. Reaction field electrostatics are used.

9.1.1.35 Test Case 35: Free Energy Difference of DMSO/DMSO* in DMSO Solvent

This simulation represents a single point in a thermodynamic integration procedure to determine the free energy difference between an excited DMSO molecule (labelled DMSO*) and the ground-state DMSO molecule. The simulation corresponds to a mixed Hamiltonian system of 512 DMSO molecules (502 representing the solvent, 10 representing the ground state) and 10 DMSO* excited

molecules. The mixing uses the error function method with $\lambda = 0.25$. The basic ensemble is provided by the Hoover NVT algorithm. The electrostatic interactions are handled by the reaction field method.

9.1.1.36 Test Case 36. Calculation of Solvent Induced Spectral Shift

In this simulation 512 DMSO molecules are simulated in the Hoover NVT ensemble and at intervals 10 DMSO molecules are substituted by DMSO* molecules in the same configuration in order to determine the instantaneous solvation energy of the DMSO*. The simulation immediately reverts back to the groundstate DMSO to continue. The electrostatic interactions are handled by the reaction field method.

9.1.1.37 Test Case 37. Calculation of Solvent Relaxation following Spectral Excitation

This is a Hoover NVT simulation of 512 DMSO molecules in which, after a fixed interval, 10 DMSO molecules are replaced by DMSO* and the subsequent simulation records the energetic response of the solvent to the excitation. After another interval, the reverse switch is enacted and the DMSO* molecules are replaced by DMSO, to determine the relaxation after quenching. The electrostatic interactions are handled by the reaction field method.

9.1.1.38 Test Case 38. Freezing of TIP4P Water

This is a metadynamics simulation of the freezing of water at 180K and 1 atmosphere pressure using a 4-centre TIP4P rigid model of the water molecule. The system consists of 512 water molecules and the ensemble is Hoover NPT. Two order parameters are used to define the structures: global potential energy and the Steinhardt Q_6 parameter. Control of the Gaussian convergence is by well-tempered dynamics.

9.1.1.39 Test Case 39: Calcite Nanoparticle Metadynamics

In this case 75 molecules of calcium carbonate in the calcite structure form a nanoparticle which is suspended in 863 water molecules represented by a flexible 3-centre TIP3P model. The temperature is 310K and pressure 1 atmosphere maintained in a Hoover NPT ensemble. The metadynamics is controlled by 6 order parameters: the global potential energy and 5 Steinhardt Q_4 parameters. Gaussian convergence is controlled by well-tempered dynamics.

9.1.2 Benchmark Cases

These represent rather larger test cases for DL_POLY Classic that are also suitable for benchmarking the code on large scale computers. They have been selected to show fairly the the capabilities and limitations of the code.

9.1.2.1 Benchmark 1

Simulation of metallic aluminium at 300K using a Sutton-Chen density dependent potential. The system is comprised of 19652 identical atoms. The simulation runs on 16 to 512 processors only.

9.1.2.2 Benchmark 2

Simulation of a 15-peptide in 1247 water molecules. This was designed as an AMBER comparison. The system consists of 3993 atoms in all and runs on 8-512 processors. It uses neutral group electrostatics and rigid bond constraints and is one of the smallest benchmarks in the set.

9.1.2.3 Benchmark 3

Simulation of the enzyme transferrin in 8102 water molecules. The simulation makes use of neutral group electrostatics and rigid bond constraints. The system is 27539 atoms and runs on 8-512 processors.

9.1.2.4 Benchmark 4

Simulation of a sodium chloride melt with Ewald sum electrostatics and a multiple timestep algorithm to enhance performance. The system is comprised of 27000 atoms and runs on 8-512 processors.

9.1.2.5 Benchmark 5

Simulation of a sodium-potassium disilicate glass. Uses Ewald sum electrostatics, a multiple timestep algorithm and a three-body valence angle potentials to support the silicate structure. It also using tabluated two-body potentials stored in the file TABLE. The system is comprised of 8640 atoms and runs on 16-512 processors.

9.1.2.6 Benchmark 6

Simulation of a potassium-valinomycin complex in 1223 water molecules using an adapted AMBER forcefield and truncated octahedral periodic boundary conditions. The system size is 3838 atoms and runs on 16-512 processors.

9.1.2.7 Benchmark 7

Simulation of gramicidin A molecule in 4012 water molecules using neutral group electrostatics. The system is comprised of 12390 atoms and runs on 8-512 processors. This example was provided by Lewis Whitehead at the University of Southampton.

9.1.2.8 Benchmark 8

Simulation of an isolated magnesium oxide microcrystal comprised of 5416 atoms originally in the shape of a truncated octahedron. Uses full coulombic potential. Runs on 16-512 processors.

9.1.2.9 Benchmark 9

Simulation of a model membrane with 196 41-unit membrane chains, 8 valinomycin molecules and 3144 water molecules using an adapted AMBER potential, multiple timestep algorithm and Ewald sum electrostatics. The system is comprised of 18866 atoms and runs on 8-512 processors.

Chapter 10

Utilities

Scope of Chapter

This chapter describes the more important utility programs and subroutines of DL_POLY Classic, found in the sub-directory *utility*.

10.1 Miscellaneous Utilities

10.1.1 Useful Macros

10.1.1.1 Macros

Macros are simple executable files containing standard unix commands. A number of the are supplied with DL_POLY and are found in the *execute* sub-directory. The available macros are as follows.

- *cleanup*
- *copy*
- *gopoly*
- *gui*
- *select*
- *store*
- *supa*

The function of each of these is described below. It is worth noting that most of these functions can be performed by the DL_POLY Classic java GUI [9].

10.1.1.2 *cleanup*

cleanup removes several standard data files from the *execute* sub-directory. It contains the unix commands:

```
#!/bin/tcsh
#
# DL_POLY utility to clean up after a program run
#
if (-e  CFGMIN) rm CFGMIN
if (-e  OUTPUT) rm OUTPUT
if (-e  RDFDAT) rm RDFDAT
if (-e  REVCON) rm REVCON
if (-e  REVIVE) rm REVIVE
if (-e  REVOLD) rm REVOLD
if (-e  STATIS) rm STATIS
if (-e  ZDNDAT) rm ZDNDAT
```

and removes the files (if present) CFGMIN, OUTPUT, REVCON, REVOLD, STATIS, REVIVE, RDFDAT and ZDNDAT. (Useful data should be stored elsewhere beforehand!)

10.1.1.3 *copy*

copy invokes the unix commands:

```
#!/bin/tcsh
#
# utility to set up data for DL_POLY continuation run
#
mv CONFIG CONFIG.OLD
mv REVCON CONFIG
mv REVIVE REVOLD
```

which collectively prepare the DL_POLY files in the *execute* sub-directory for the continuation of a simulation. It is always a good idea to store these files elsewhere in addition to using this macro.

10.1.1.4 *gopoly*

gopoly is used to submit a DL_POLY job to the Daresbury Intel Xeon cluster and takes a form similar for batch processing on many parallel machines. The following is for an 8 processor job:

```
#!/bin/bash
#
# The parallel environment to run in and the number of nodes
#$ -pe mpich 2
#
# Run from the directory the job was submitted from
#$ -cwd
#
# Export all environment variables from submission shell
# to the job
#$ -V
#
# Merge stderr and stdout streams
#$ -j yes
#
# What to name the output
#$ -o GOPOLY.$JOB_ID

# How many processors per node
PPN=4

# The location of the binary to run
binary=/home/user/dl_poly_2.20/execute/DLPOLY.X

# Create the machinefile
sed s/$/:4/ $HOME/.mpich/mpich_hosts.$JOB_ID > \
$HOME/.mpich/ndfile.$JOB_ID

# Do it!
mpirun \
-np $((NSLOTS*PPN)) \
```

```
-machinefile $HOME/.mpich/ndfile.$JOB_ID \
$binary
```

Normally the job is submitted by the unix command:

qsub gopoly

where *qsub* is a local command for submission to the Xeon cluster. The number of required nodes and the job time are indicated in the above script.

10.1.1.5 *gui*

gui is a macro that starts up the DL_POLY Classic Java GUI. It invokes the following unix commands:

```
java -jar ../java/GUI.jar
```

In other words the macro invokes the Java Virtual Machine which executes the instructions in the Java archive file GUI.jar, which is stored in the *java* subdirectory of DL_POLY Classic. (Note: Java 1.3.0 or a higher version is required to run the GUI.)

10.1.1.6 *select*

select is a macro enabling easy selection of one of the test cases. It invokes the unix commands:

```
#!/bin/tcsh
#
# DL_POLY utility to gather test data files for program run
#
cp ../data/TEST$1/$2/CONTROL CONTROL
cp ../data/TEST$1/$2/FIELD   FIELD
cp ../data/TEST$1/$2/CONFIG  CONFIG
if (-e  ../data/TEST$1/$2/TABLE)then
  cp ../data/TEST$1/$2/TABLE   TABLE
else if (-e  ../data/TEST$1/$2/TABEAM)then
  cp ../data/TEST$1/$2/TABEAM  TABEAM
endif
```

select requires two arguments to be specified:

select n a

where n is the (integer) test case number, which ranges from 1 to 20 and a is the character string LF, VV, RB or CB according to which algorithm leapfrog (LF), velocity Verlet (VV), (RB) rigid body minimisation or (CB) constraint bond minimisation is required.

This macro sets up the required input files in the *execute* sub-directory to run the n-th test case.

10.1.1.7 *store*

The *store* macro provides a convenient way of moving data back from the *execute* sub-directory to the *data* sub-directory. It invokes the unix commands:

```
#!/bin/tcsh
#
# DL_POLY utility to archive I/O files to the data directory
#
if !(-e ../data/TEST$1) then
  mkdir ../data/TEST$1
endif
if !(-e ../data/TEST$1/$2) then
  mkdir ../data/TEST$1/$2
endif
mv CONTROL ../data/TEST$1/$2/CONTROL
mv FIELD ../data/TEST$1/$2/FIELD
mv CONFIG ../data/TEST$1/$2/CONFIG
mv OUTPUT ../data/TEST$1/$2/OUTPUT
mv REVIVE ../data/TEST$1/$2/REVIVE
mv REVCON ../data/TEST$1/$2/REVCON
if (-e TABLE) then
  mv TABLE ../data/TEST$1/$2/TABLE
endif
if (-e TABEAM) then
  mv TABEAM ../data/TEST$1/$2/TABEAM
endif
if (-e STATIS) then
  mv STATIS ../data/TEST$1/$2/STATIS
endif
if (-e RDFDAT) then
  mv RDFDAT ../data/TEST$1/$2/RDFDAT
endif
if (-e ZDNDAT) then
  mv ZDNDAT ../data/TEST$1/$2/ZDNDAT
endif
if (-e CFGMIN) then
  mv CFGMIN ../data/TEST$1/$2/CFGMIN
endif
```

which first creates a new DL_POLY *data/TEST..* if necessary sub-directory and then moves the standard DL_POLY output data files into it.

store requires two arguments:

store n a

where *n* is a unique string or number to label the output data in the *data/TESTn* sub-directory and *a* is the character string LF, VV, RB or CB according to which algorithm leapfrog (LF), velocity Verlet (VV), (RB) rigid body minimisation or (CB) constraint bond minimisation has been performed.

10.1.1.8 *supa*

The *supa* macro provides a convenient way of running the DL_POLY test cases in batch mode. It is currently structured to submit batch jobs to the Daresbury Xeon cluster, but can easily be adapted for other machines where batch queuing is possible. The key statement in this context in the 'qsub' commmand which submits the *gopoly* script described above. This statement may be replaced by the equivalent batch queuing command for your machine. The text of *supa* is given below.

```
#!/bin/tcsh
#
# DL_POLY script to run multiple test cases
# note use of qsub in job submission - may
# need replacing
#

set n=$1
set m=$2
set TYPE="LF VV CB RB"

while ($n <= $m)
  if !(-e  TEST$n) mkdir TEST$n
  cd TEST$n
  echo TEST$n
  foreach typ ($TYPE)
    if (-e ../../data/TEST$n/$typ ) then
      if !(-e  $typ) mkdir $typ
      cd $typ
      cp ../../../data/TEST$n/$typ/CONTROL .
      cp ../../../data/TEST$n/$typ/CONFIG .
      cp ../../../data/TEST$n/$typ/FIELD .
      if (-e  ../../../data/TEST$n/$typ/TABLE) \
      cp ../../../data/TEST$n/$typ/TABLE .
      if(-e  ../../../data/TEST$n/$typ/TABEAM) \
      cp ../../../data/TEST$n/$typ/TABEAM .
      qsub ../../gopoly
      cd ../
    endif
  end
  cd ../
  set n=‘expr $n + 1‘
end
```

This macro creates working *TEST* directories in the *execute* sub-directory; one for each test case invoked. Appropriate sub-directories of these are created for leapfrog (LF), velocity Verlet(VV), rigid body minimisation (RB) and constraint bond minimisation (CB). Note that *supa* must be run from the *execute* sub-directory.

supa requires two arguments:

supa n m

where n and m are integers defining the first and last test case to be run.

Bibliography

[1] Smith, W., and Forester, T., 1996, *J. Molec. Graphics*, **14**, 136. 3

[2] Smith, W., 1987, *Molecular Graphics*, **5**, 71. 3

[3] Finnis, M. W., and Sinclair, J. E., 1984, *Philos. Mag. A*, **50**, 45. 4, 34, 35, 118, 119

[4] Johnson, R. A., 1989, *Phys. Rev. B*, **39**, 12556. 4, 41

[5] Tersoff, J., 1989, *Phys. Rev. B*, **39**, 5566. 4, 31, 120, 121

[6] van Gunsteren, W. F., and Berendsen, H. J. C. 1987, *Groningen Molecular Simulation (GROMOS) Library Manual*. BIOMOS, Nijenborgh, 9747 Ag Groningen, The Netherlands. Standard GROMOS reference. 4, 13

[7] Mayo, S., Olafson, B., and Goddard, W., 1990, *J. Phys. Chem.*, **94**, 8897. 4, 13, 30, 31, 119, 187, 198

[8] Weiner, S. J., Kollman, P. A., Nguyen, D. T., and Case, D. A., 1986, *J. Comp. Chem.*, **7**, 230. 4, 13

[9] Smith, W., 2003, *Daresbury Laboratory*. 4, 9, 83, 105, 226, 234

[10] Smith, W., and Forester, T. R., 1994, *Comput. Phys. Commun.*, **79**, 52. 5

[11] Smith, W., and Forester, T. R., 1994, *Comput. Phys. Commun.*, **79**, 63. 5, 57, 59

[12] Allen, M. P., and Tildesley, D. J. 1989, *Computer Simulation of Liquids*. Oxford: Clarendon Press. 5, 14, 46, 54, 57, 59, 75, 76

[13] Ryckaert, J. P., Ciccotti, G., and Berendsen, H. J. C., 1977, *J. Comput. Phys.*, **23**, 327. 5, 57, 75

[14] Andersen, H. C., 1983, *J. Comput. Phys.*, **52**, 24. 5, 58

[15] Fincham, D., 1992, *Molecular Simulation*, **8**, 165. 5, 55, 69

[16] Miller, T., Eleftheriou, M., Pattnaik, P., Ndirango, A., Newns, D., and Martyna, G., 2002, *J. Chem. Phys.*, **116**, 8649. 5, 56, 69, 70

[17] Forester, T., and Smith, W., 1998, *J Computational Chemistry*, **19**, 102. 5, 55, 56, 71

[18] Martyna, G., Tuckerman, M., Tobias, D., and Klein, M., 1996, *Molec. Phys.*, **87**, 1117. 5, 60, 70

[19] Evans, D. J., and Morriss, G. P., 1984, *Computer Physics Reports*, **1**, 297. 5, 55, 56, 59

[20] Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W., DiNola, A., and Haak, J. R., 1984, *J. Chem. Phys.*, **81**, 3684. 5, 55, 56, 59

[21] Hoover, W. G., 1985, *Phys. Rev.*, **A31**, 1695. 5, 55, 56, 59

[22] Jorgensen, W. L., Madura, J. D., and Swenson, C. J., 1984, *J. Amer. Chem. Soc*, **106**, 6638. 13, 187

[23] Brode, S., and Ahlrichs, R., 1986, *Comput. Phys. Commun.*, **42**, 41. 14, 76

[24] Hockney, R. W., and Eastwood, J. W. 1981, *Computer Simulation Using Particles*. McGraw-Hill International. 14, 15, 78

[25] Warner, H. R. J., 1972, *ind. Eng. Chem. Fundam.*, **11**, 379. 16

[26] Bird, R. B. e. a. 1977, *Dynamics of Polymeric Liquids*, volume 1 and 2. Wiley, New York. 16

[27] Grest, G. S., and Kremer, K., 1986, *Phys. Rev. A*, **33**, 3628. 16

[28] Vessal, B., 1994, *J. Non-Cryst. Solids*, **177**, 103. 18, 20, 31, 113, 119

[29] Smith, W., Greaves, G. N., and Gillan, M. J., 1995, *J. Chem. Phys.*, **103**, 3091. 18, 20, 31, 113, 119

[30] Smith, W., 1993, *CCP5 Information Quarterly*, **39**, 14. 19, 22, 25

[31] Rohl, A. L., Wright, K., and Gale, J. D., 2003, *Amer. Mineralogist*, **88**, 921. 25

[32] Clarke, J. H. R., Smith, W., and Woodcock, L. V., 1986, *J. Chem. Phys.*, **84**, 2290. 28, 118

[33] Weeks, J. D., Chandler, D., and Anderson, H. C., 1971, *J. Chem. Phys.*, **54**, 5237. 29

[34] Eastwood, J. W., Hockney, R. W., and Lawrence, D. N., 1980, *Comput. Phys. Commun.*, **19**, 215. 31, 33, 34

[35] Daw, M. S., and Baskes, M. I., 1984, *Phys. Rev. B*, **29**, 6443. 34, 118

[36] Foiles, S. M., Baskes, M. I., and Daw, M. S., 1986, *Chem. Phys. Lett.*, **33**, 7983. 34, 118

[37] J., F., 1952, *Philos. Mag.*, **43**, 153. 34

[38] Sutton, A. P., and Chen, J., 1990, *Philos. Mag. Lett.*, **61**, 139. 35, 78, 119

[39] Rafii-Tabar, H., and Sutton, A. P., 1991, *Philos. Mag. Lett.*, **63**, 217. 35, 41, 119

[40] Todd, B., and Lynden-Bell, R., 1993, *Surf. Science*, **281**, 191. 35

[41] Cleri, F., and Rosato, F., 1993, *Phys. Rev. B*, **48**, 22. 35, 119

[42] Wolf, D., Keblinski, P., Phillpot, S., and Eggebrecht, J., 1999, *J. Chem. Phys.*, **110**, 8255. 45

[43] Fennell, C., and Gezelter, J., 2006, *J. Chem. Phys.*, **124**, 234104. 45

[44] Fuchs, K., 1935, *Proc. R. Soc., A*, **151**, 585. 47

[45] Smith, W., and Fincham, D., 1993, *Molecular Simulation*, **10**, 67. 47, 74, 75, 79, 90

[46] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G., 1995, *J. Chem. Phys.*, **103**, 8577. 48

[47] Hautman, J., and Klein, M. L., 1992, *Molec. Phys.*, **75**, 379. 50, 250

[48] Neumann, M., 1985, *J. Chem. Phys.*, **82**, 5663. 52

[49] Fincham, D., and Mitchell, P. J., 1993, *J. Phys. Condens. Matter*, **5**, 1031. 53

[50] Lindan, P. J. D., and Gillan, M. J., 1993, *J. Phys. Condens. Matter*, **5**, 1019. 54

[51] McCammon, J. A., and Harvey, S. C. 1987, *Dynamics of Proteins and Nucleic Acids*. Cambridge: University Press. 59

[52] Brown, D., and Clarke, J. H. R., 1984, *Molec. Phys.*, **51**, 1243. 62

[53] Melchionna, S., Ciccotti, G., and Holian, B. L., 1993, *Molec. Phys.*, **78**, 533. 63

[54] Tildesley, D. J., Streett, W. B., and Saville, G., l978, *Molec. Phys*, **35**, 639. 74

[55] Tildesley, D. J., and Streett, W. B. Multiple time step methods and an improved potential function for molecular dynamics simulations of molecular liquids. In Lykos, P., editor, *Computer Modelling of Matter*. ACS Symposium Series No. 86, 1978. 74

[56] Forester, T., and Smith, W., 1994, *Molecular Simulation*, **13**, 195. 74

[57] Smith, W., 1991, *Comput. Phys. Commun.*, **62**, 229. 74, 75, 78

[58] Smith, W., 1993, *Theoretica. Chim. Acta.*, **84**, 385. 74, 75, 76

[59] Smith, W., 1992, *Comput. Phys. Commun.*, **67**, 392. 75, 78

[60] Vessal, B., Amini, M., Leslie, M., and Catlow, C. R. A., 1990, *Molecular Simulation*, **5**, 1. 78, 201

[61] Shewchuk, J. R. August 4, 1994, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, Edition 1 1/4*. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. 88

[62] Voter, A., 1997, *J. Chem. Phys.*, **106**, 4665. 138, 140, 141

[63] Sorensen, M., and Voter, A., 2000, *J. Chem. Phys.*, **112**, 9599. 138, 147, 149, 150

[64] Hamelberg, D., Mongan, J., and McCammon, J. A., 2004, *J. Chem. Phys.*, **120**, 11919. 138, 141, 142, 143

[65] Henkelman, G., and Jonsson, H., 2000, *J. Chem. Phys.*, **113**, 9978. 139, 140

[66] Rahman, J. A., and Tully, J. C., 2002, *J. Chem. Phys.*, **116**, 8750. 142

[67] Laio, A., and Parrinello, M., 2002, *Proc. Natl. Acad. Sci.*, **99**, 12562. 176

[68] Quigley, D., and Rodger, P., 2009, *Molecular Simulation*, **35**, 613. 176, 177, 178, 179, 180, 182, 183

[69] Laio, A., Rordiguez-Fortea, A., Gervasio, F. L., Ceccarelli, M., and Parrinello, M., 2005, *J. Phys. Chem. B*, **109**, 6714. 177, 180

[70] Peters, B., and Trout, B. L., 2006, *J. Chem. Phys.*, **125**, 054108. 177

[71] Donadio, D., Raiteri, P., and Parrinello, M., 2005, *J. Phys. Chem. B*, **109**, 5421. 177

[72] Steinhardt, P. J., Nelson, D. R., and Ronchetti, M., 1983, *Phys. Rev. B*, **28**, 784. 177, 178

[73] 177, 179

[74] 184

[75] Lewis, G. V., and Catlow, C. R. A., 1985, *Solid State Phys. C*, **18**, 1149. 187, 200

[76] Bush, T. S., Gale, J. D., Catlow, C. R. A., and Battle, D., 1994, *J. Mater. Chem.*, **4**, 831. 187, 200

[77] Vashishta, A., and Rahman, A., 1978, *Phys. Rev. Letters*, **40**, 1337. 201

Appendix A

The DL_POLY Classic Makefile

```
# Master makefile for DL_POLY Classic
# Author: W. Smith January Dec 2010
#
#=====================================================================
# Define default settings
#=====================================================================

BINROOT = ../execute
CC  = gcc
EX = DLPOLY.X
EXE = $(BINROOT)/$(EX)
FC=undefined
SHELL=/bin/sh
TYPE=par

#=====================================================================
# Define object files

OBJ_MOD = parse_module.o setup_module.o error_module.o \
        site_module.o config_module.o pair_module.o utility_module.o \
        metafreeze_module.o solvation_module.o tether_module.o \
        vdw_module.o property_module.o rigid_body_module.o \
        angles_module.o bonds_module.o shake_module.o \
        inversion_module.o dihedral_module.o core_shell_module.o \
        exclude_module.o ewald_module.o coulomb_module.o\
        external_field_module.o four_body_module.o \
        hkewald_module.o metal_module.o ensemble_tools_module.o \
        temp_scalers_module.o three_body_module.o spme_module.o \
        tersoff_module.o neu_coul_module.o \
        nlist_builders_module.o forces_module.o \
        lf_motion_module.o lf_rotation1_module.o \
        lf_rotation2_module.o vv_motion_module.o \
        vv_rotation1_module.o vv_rotation2_module.o \
        pmf_module.o integrator_module.o optimiser_module.o \
        hyper_dynamics_module.o driver_module.o \
        define_system_module.o
```

```
OBJ_SRC = dlpoly.o

OBJ_PAR = basic_comms.o merge_tools.o pass_tools.o

#=====================================================================
# Define targets
all:
        @echo "Error - please specify a target machine!"
        @echo "Permissible targets for this Makefile are:"
        @echo "                                              "
        @echo "gfortran                    (parallel)"
        @echo "woodcrest                   (parallel)"
        @echo "         "
        @echo "Please examine Makefile for details"

# system specific targets follow :

#================== GNU Fortran, MPI version ============================
gfortran:
        $(MAKE) FC="mpif90" LD="mpif90 -o" \
        LDFLAGS="-O2 -ffast-math" \
        FFLAGS="-c -O2 -ffast-math" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#================ Woodcrest ======================================
woodcrest:
        $(MAKE) LD="mpif90 -o" LDFLAGS="" \
        FC=mpif90 FFLAGS="-c -O3" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#=====================================================================
# Default code for parallel (MPI) execution

par: check $(OBJ_MOD) $(OBJ_PAR) $(OBJ_SRC)
        $(LD) $(EX) $(LDFLAGS) $(OBJ_MOD) $(OBJ_PAR) $(OBJ_SRC)
        mv $(EX) $(EXE)

#=====================================================================
# Check that a machine has been specified
check:
        @if test $(FC) = "undefined";\
        then echo "You must specify a target machine!"; \
        exit 99;\
        fi

#=====================================================================
# Clean up the source directory
clean:
```

```
        rm -f $(OBJ_MOD) $(OBJ_PAR) $(OBJ_SRC) *.mod

#=======================================================================
# Declare dependencies
.f.o:
        $(FC) $(FFLAGS) $*.f
.c.o:
        $(CC) -c $*.c

#=======================================================================
# Declare dependency on module files

$(OBJ_SRC): $(OBJ_MOD)
```

Appendix B

Periodic Boundary Conditions in DL_POLY Classic

Introduction

DL_POLY Classic is designed to accommodate a number of different periodic boundary conditions, which are defined by the shape and size of the simulation cell. Briefly, these are as follows (which also indicates the IMCON flag defining the simulation cell type in the CONFIG File - see 4.1.2):

1. None e.g. isolated polymer in space. (IMCON=0).
2. Cubic periodic boundaries.(IMCON=1).
3. Orthorhombic periodic boundaries.(IMCON=2).
4. Parallelepiped periodic boundaries.(IMCON=3).
5. Truncated octahedral periodic boundaries. (IMCON=4).
6. Rhombic dodecahedral periodic boundaries. (IMCON=5).
7. Slab (X,Y periodic, Z nonperiodic). (IMCON=6).
8. Hexagonal prism periodic boundaries. (IMCON=7).

We shall now look at each of these in more detail. Note that in all cases the cell vectors and the positions of the atoms in the cell are to be specified in Angstroms (Å).

No periodic boundary (IMCON=0)

Simulations requiring no periodic boundaries are best suited to *in vacuuo* simulations, such as the conformational study of an isolated polymer molecule. This boundary condition is not recommended for studies in a solvent, since evaporation is likely to be a problem.

Note this boundary condition cannot be used with the Ewald summation method.

Cubic periodic boundaries (IMCON=1)

The cubic MD cell is perhaps the most commonly used in simulation and has the advantage of great simplicity. In DL_POLY Classic the cell is defined with the principle axes passing through the centres of the faces. Thus for a cube with sidelength D, the cell vectors appearing in the CONFIG file should be: (D,0,0); (0,D,0); (0,0,D). Note the origin of the atomic coordinates is the centre of the cell.

The cubic boundary condition can be used with the Ewald summation method.

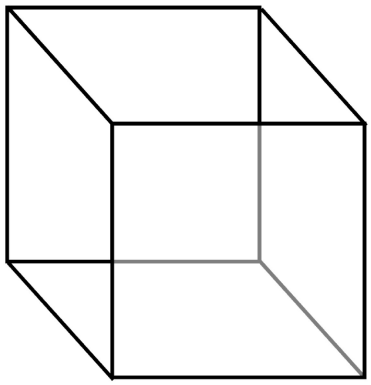

Figure B.1: The cubic MD cell.

Orthorhombic periodic boundaries (IMCON=2)

The orthorhombic cell is also a common periodic boundary, which closely resembles the cubic cell in use. In DL_POLY Classic the cell is defined with principle axes passing through the centres of the faces. For an orthorhombic cell with sidelengths D (in X-direction), E (in Y-direction) and F (in Z-direction), the cell vectors appearing in the CONFIG file should be: (D,0,0); (0,E,0); (0,0,F). Note the origin of the atomic coordinates is the centre of the cell.

The orthorhombic boundary condition can be used with the Ewald summation method.

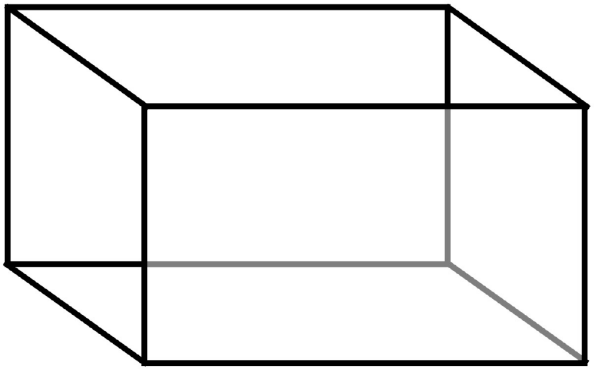

Figure B.2: The orthorhomic MD cell.

Parallelepiped periodic boundaries (IMCON=3)

The parallelepiped (e.g. monoclinic or triclinic) cell is generally used in simulations of crystalline materials, where its shape and dimension is commensurate with the unit cell of the crystal. Thus for a unit cell specified by three principal vectors $\underline{a}$, $\underline{b}$, $\underline{c}$, the MD cell is defined in the DL_POLY Classic CONFIG file by the vectors (La_1,La_2,La_3), (Mb_1,Mb_2,Mb_3), (Nc_1,Mc_2,Nc_3), in which L,M,N are integers, reflecting the multiplication of the unit cell in each principal direction. Note that the atomic coordinate origin is the centre of the MD cell.

The parallelepiped boundary condition can be used with the Ewald summation method.

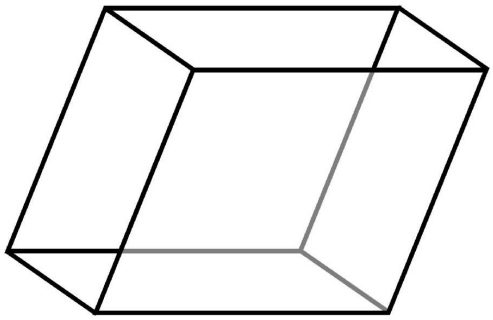

Figure B.3: The parallelepiped MD cell.

Truncated octahedral boundaries (IMCON=4)

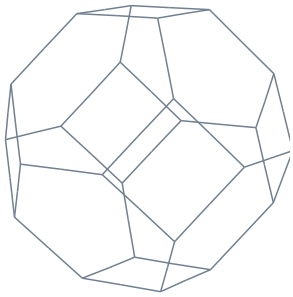

Figure B.4: The truncated octahedral MD cell.

This is one of the more unusual MD cells available in DL_POLY, but it has the advantage of being more nearly spherical than most other MD cells. This means it can accommodate a larger spherical cutoff for a given number of atoms, which leads to greater efficiency. This can be very useful when simulating (for example) a large molecule in solution, where fewer solvent molecules are required for a given simulation cell width.

The principal axes of the truncated octahedron (see figure) pass through the centres of the square faces, and the width of the cell, measured from square face to square face along a principal axis defines the width D of the cell. From this, the cell vectors required in the DL_POLY Classic CONFIG file are simply: (D,0,0), (0,D,0), (0,0,D). These are also the cell vectors defining the enscribing cube, which posseses twice the volume of the truncated octahedral cell. Once again, the atomic positions are defined with respect to the cell centre.

The truncated octahedron can be used with the Ewald summation method.

Rhombic dodecahedral boundaries (IMCON=5)

This is another unusual MD cell (see figure), but which possesses similar advantages to the truncated octahedron, but with a slightly greater efficiency in its use of the cell volume (the ratio is about

74% to 68%).

The principal axis in the X-direction of the rhombic dodecahedron passes through the centre of the cell and the centre of a rhombic face. The Y-axis does likewise, but is set at 90 degrees to the X-axis. The Z-axis completes the orthonormal set and passes through a vertex where four faces meet. If the width D of the cell is defined as the perpendicular distance between two opposite faces, the cell vectors required for the DL_POLY Classic CONFIG file are: (D,0,0), (0,D,0), (0,0,$\sqrt{2}$D).These also define the enscribing orthorhombic cell, which has twice the MD cell volume. In DL_POLY Classic the centre of the cell is also the origin of the atomic coordinates.

The rhombic dodecahedron can be used with the Ewald summation method.

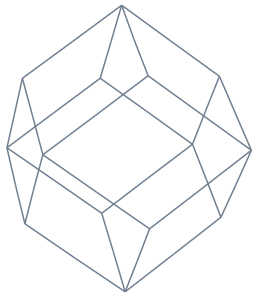

Figure B.5: The rhombic dodecahedral MD cell.

Slab boundary conditions (IMCON=6)

Slab boundaries are periodic in the X- and Y-directions, but not in the Z-direction. They are particularly useful for simulating surfaces. The periodic cell in the XY plane can be any parallelogram. The origin of the X,Y atomic coordinates lies on an axis perpendicular to the centre of the parallelogram. The origin of the Z coordinate is where the user specifies it, but at or near the surface is recommended.

If the XY parallelogram is defined by vectors $\underline{A}$ and $\underline{B}$, the vectors required in the CONFIG file are: (A_1,A_2,0), (B_1,B_2,0), (0,0,D), where D is any real number (including zero). If D is nonzero, it will be used by DL_POLY to help determine a 'working volume' for the system. This is needed to help calculate RDFs etc. (The working value of D is in fact taken as one of: 3×cutoff; or 2×max abs(Z coordinate)+cutoff; or the user specified D, whichever is the larger.)

Note that the standard Ewald sum cannot be used with this boundary condition. DL_POLY Classic switches automatically to the Hautman-Klein-Ewald method instead [47].

The surface in a system with charges can also be modelled with DL_POLY Classic if periodicity is allowed in the Z-direction. In this case slabs of ions well-separated by vacuum zones in the Z-direction can be handled with IMCON=2 or 3.

Hexagonal prism boundaries (IMCON=7)

In this case the Z-axis lies along a line joining the centres of the hexagonal faces. The Y-axis is perpendicular to this and passes through the centre of one of the faces. The X-axis completes the orthonormal set and passes through the centre of an edge that is parallel to the Z-axis. (Note: It is important to get this convention right!) The origin of the atomic coordinates is the centre of the

cell. If the length of one of the hexagon edges is D, the cell vectors required in the CONFIG file are: (3D,0,0), (0,$\sqrt{3}$D,0), (0,0,H), where H is the prism height (the distance between hexagonal faces). The orthorhombic cell also defined by these vectors enscribes the hexagonal prism and possesses twice the volume, but the height and the centre are the same.

The Ewald summation method may be used with this periodic boundary condition.

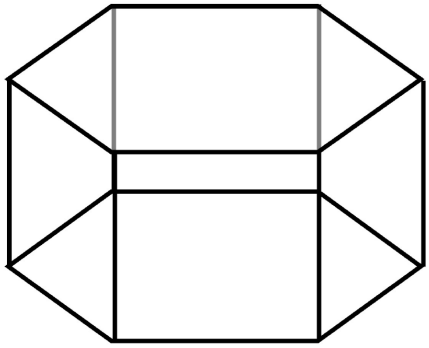

Figure B.6: The hexagonal MD cell.

This MD cell is particularly suitable for simulating strands or fibres (i.e. systems with a pronounced anisotropy in the Z-direction), such as DNA strands in solution, or stretched polymer chains.

Appendix C

Error Messages and User Action

Introduction

In this appendix we document the error messages encoded in DL_POLY Classic and the recommended user action. The correct response is described as the **standard user response** in the approriate sections below, to which the user should refer before acting on the error encountered.

The reader should also be aware that some of the error messages listed below may be either disabled in, or absent from, the installed version of DL_POLY Classic. Disabled messages generally apply to older releases of the code, while absent messages apply to newer versions of the code and will not usually apply to previous releases. They are all included for completeness. Note that the wording of some of the messages may also have changed over time, usually to provide more specific information. The most recent wording appears below.

DL_POLY Classic incorporates FORTRAN 90 dynamic array allocation to set the array sizes at run time. It is not foolproof however. Sometimes an estimate of the required array sizes is difficult to obtain and the calculated value may be too small. For this reason DL_POLY Classic retains a number of array dimension checks and will terminate when an array bound error occurs.

When a dimension error occurs, the **standard user response** is to edit the DL_POLY Classic subroutine PARSET.F. Locate where the variable defining the array dimension is fixed and increase accordingly. To do this you should make use of the dimension information that DL_POLY Classic prints in the OUTPUT file prior to termination. If no information is supplied, simply doubling the size of the variable will usually do the trick. If the variable concerned is defined in one of the support subroutines CFGSCAN.F, FLDSCAN.F, CONSCAN.F you will need to insert a new line in PARSET.F to redefine it - after the relevant subroutine has been called! Finally the code must be recompiled, but in this case it will be necessary only to recompile PARSET.F and not the whole code.

The DL_POLY Classic Error Messages

Message 3: error - unknown directive found in CONTROL file

This error most likely arises when a directive is misspelt.

Action:
Locate incorrect directive in CONTROL file and replace.

Message 4: error - unknown directive found in FIELD file

This error most likely arises when a directive is misspelt or is encountered in an incorrect location in the FIELD file, which can happen if too few or too many data records are included.

Action:
Locate the erroneous directive in the FIELD file and correct error.

Message 5: error - unknown energy unit requested

The DL_POLY Classic FIELD file permits a choice of units for input of energy parameters. These may be: electron volts (**ev**); kilocalories (**kcal**); kilojoules (**kj**); or the DL_POLY Classic internal units (10 J mol^{-1}) (**internal**). There is no default value. Failure to specify any of these correctly, or reference to other energy units, will result in this error message. See documentation of the FIELD file.

Action:
Correct energy keyword on **units** directive in FIELD file and resubmit.

Message 6: error - energy unit not specified

A **units** directive is mandatory in the FIELD file. This error indicates that DL_POLY Classic has failed to find the required record.

Action:
Add **units** directive to FIELD file and resubmit.

Message 7: error - energy unit respecified

DL_POLY Classic expects only one **units** directive in the FIELD file. This error results if it encounters another - implying an ambiguity in units.

Action:
Locate extra **units** directive in FIELD file and remove.

Message 8: error - time step not specified

DL_POLY Classic requires a **timestep** directive in the CONTROL file. This error results if none is encountered.

Action:
Insert**timestep** directive in CONTROL file with an appropriate numerical value.

Message 10: error - too many molecule types specified

DL_POLY Classic has a set limit on the number of kinds of molecules it will handle in any simulation (this is not the same as the number of molecules). If this permitted maximum is exceeded, the program terminates. The error arises when the **molecules** directive in the FIELD file specifes too large a number.

Action:
Standard user response. Fix parameter `mxtmls`.

Message 11: error - duplicate molecule directive in FIELD file

The number of different types of molecules in a simulation should only be specified once. If DL_POLY Classic encounters more than one **molecules** directive, it will terminate execution.

Action:
Locate the extra **molecule** directive in the FIELD file and remove.

Message 12: error - unknown molecule directive in FIELD file

Once DL_POLY Classic encounters the **molecules** directive in the FIELD file, it assumes the following records will supply data describing the intramolecular force field. It does not then expect to encounter directives not related to these data. This error message results if it encounters a unrelated directive. The most probable cause is incomplete specification of the data (e.g. when the **finish** directive has been omitted.)

Action:
Check the molecular data entries in the FIELD file and correct.

Message 13: error - molecule species not yet specified

This error arises when DL_POLY Classic encounters non-bonded force data in the FIELD file, *before* the molecular species have been specified. Under these circumstances it cannot assign the data correctly, and therefore terminates.

Action:
Make sure the molecular data appears before the non-bonded forces data in the FIELD file and resubmit.

Message 14: error - too many unique atom types specified

This error arises when DL_POLY Classic scans the FIELD file and discovers that there are too many different types of atoms in the system (i.e. the number of unique atom types exceeds the `mxsvdw` parameter.

Action:
Standard user response. Fix parameter `mxsvdw`.

Message 15: error - duplicate pair potential specified

In processing the FIELD file, DL_POLY Classic keeps a record of the specified short range pair potentials as they are read in. If it detects that a given pair potential has been specified before, no attempt at a resolution of the ambiguity is made and this error message results. See specification of FIELD file.

Action:
Locate the duplication in the FIELD file and rectify.

Message 16: error - strange exit from FIELD file processing

This should never happen! However one remote possibility is that there are more than 10,000 directives in the FIELD file! It simply means that DL_POLY Classic has ceased processing the

FIELD data, but has not reached the end of the file or encountered a **close** directive. Probable cause: corruption of the DL_POLY Classic executable or of the FIELD file. We would be interested to hear of other reasons!

Action:
Recompile the program or recreate the FIELD file. If neither of these works, send the problem to us.

Message 17: error - strange exit from CONTROL file processing

See notes on message 16 above.

Message 18: error - duplicate 3-body potential specified

DL_POLY Classic has encountered a repeat specification of a 3-body potential in the FIELD file.

Action:
Locate the duplicate entry, remove and resubmit job.

Message 19: error - duplicate 4-body potential specified

A 4-body potential has been duplicated in the FIELD file.

Action:
Locate the duplicated 4-body potential and remove. Resubmit job.

Message 20: error - too many molecule sites specified

DL_POLY Classic has a fixed limit on the number of unique molecular sites in any given simulation. If this limit is exceeded, the program terminates.

Action:
Standard user response. Fix parameter `mxsite`.

Message 21: error - duplicate tersoff potential specified

The user has defined more than one Tersoff potential for a given pair of atoms types.
Action:
Locate the duplication in the FIELD file and correct.

Message 22: error - unsuitable radial increment in TABLE file

This arises when the tabulated potentials presented in the TABLE file have an increment that is greater than that used to define the other potentials in the simulation. Ideally the increment should be $r_cut/(mxgrid-4)$, where r_cut is the potential cutoff for the short range potentials and `mxgrid` is the parameter defining the length of the interpolation arrays. An increment less than this is permissible however.

Action:
The tables must be recalculated with an appropriate increment.

Message 23: error - incompatible FIELD and TABLE file potentials

This error arises when the specification of the short range potentials is different in the FIELD and TABLE files. This usually means that the order of specification of the potentials is different. When DL_POLY Classic finds a change in the order of specification, it assumes that the user has forgotten to enter one.

Action:
Check the FIELD and TABLE files. Make sure that you correctly specify the pair potentials in the FIELD file, indicating which ones are to be presented in the TABLE file. Then check the TABLE file to make sure all the tabulated potentials are present in the order the FIELD file indicates.

Message 24: error - end of file encountered in TABLE file

This means the TABLE file is incomplete in some way: either by having too few potentials included, or the number of data points is incorrect.

Action:
Examine the TABLE file contents and regenerate it if it appears to be incomplete. If it look intact, check that the number of data points specified is what DL_POLY Classic is expecting.

Message 25: error - wrong atom type found in CONFIG file

On reading the input file CONFIG, DL_POLY Classic performs a check to ensure that the atoms specified in the configuration provided are compatible with the corresponding FIELD file. This message results if they are not.

Action:
The possibility exists that one or both of the CONFIG or FIELD files has incorrectly specified the atoms in the system. The user must locate the ambiguity, using the data printed in the OUTPUT file as a guide, and make the appropriate alteration.

Message 26: error - cutoff smaller than EAM potential range

DL_POLY Classic has detected an inconsistency in the definition of the EAM potential, namely that the user is not using the correct potential range.
Action:
Look up the correct range for this potential and adjust the DL_POLY cutoff accordingly.

Message 27: error - incompatible FIELD and TABEAM file potentials

The user has (or has not) specified a set of EAM potentials in the FIELD file which are not (or are) available in the TABEAM file.
Action:
Examine the FIELD file. Make sure you have correctly specified the EAM potentials. Check that these appear in the TABEAM file if required.

Message 28: error - transfer buffer too small in mettab

The number of points specifying an EAM potential in the TABEAM file exceeds the default buffer size in METTAB.F.

Action:
Reset the `mxbuff` parameter in PARSET.F subroutine to accommodate the required array length and recompile.

Message 29: error - end of file encountered in TABEAM file

DL_POLY Classic has reached the end of the TABEAM file without finding all the data it expects.
Action:
Either the TABEAM file is incomplete or it is improperly defined. Check the structure and content of the file with the TABEAM file specification in the manual and fix the error.

Message 30: error - too many chemical bonds specified

DL_POLY Classic sets a limit on the number of chemical bond potentials that can be specified in the FIELD file. Termination results if this number is exceeded. See FIELD file documentation. Do not confuse this error with that described by message 31 (below).

Action:
Standard user response. Fix parameter `mxtbnd`.

Message 31: error - too many chemical bonds in system

DL_POLY Classic sets a limit on the number of chemical bond potentials in the simulated system as a whole. (This number is a combination of the number of molecules and the number of bonds per molecule, divided by the number of processing nodes.) Termination results if this number is exceeded. Do not confuse this error with that described by message 30 (above).

Action:
Standard user response. Fix the parameter `mxbond`.

Message 32: error - integer array memory allocation failure

DL_POLY Classic has failed to allocate sufficient memory to accommodate one or more of the integer arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 33: error - real array memory allocation failure

DL_POLY Classic has failed to allocate sufficient memory to accommodate one or more of the real arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 34: error - character array memory allocation failure

DL_POLY Classic has failed to allocate sufficient memory to accommodate one or more of the character arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 35: error - logical array memory allocation failure

DL_POLY Classic has failed to allocate sufficient memory to accommodate one or more of the logical arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 36: error - failed fmet array allocation in mettab

DL_POLY Classic is unable to allocate the `fmet` array in the definition of an EAM potential.
Action:
Most probable cause is working too near the memory limit for the machine. Try using more processors to free up some memory. Check the TABEAM file in case the data are incorrectly specified.

Message 40: error - too many bond constraints specified

DL_POLY Classic sets a limit on the number of bond constraints that can be specified in the FIELD file. Termination results if this number is exceeded. See FIELD file documentation. Do not confuse this error with that described by message 41 (below).

Action:
Standard user response. Fix the parameter `mxtcon`.

Message 41: error - too many bond constraints in system

DL_POLY Classic sets a limit on the number of bond constraints in the simulated system as a whole. (This number is a combination of the number of molecules and the number of per molecule, divided by the number of processing nodes.) Termination results if this number is exceeded. Do not confuse this error with that described by message 40 (above).

Action:
Standard user response. Fix the parameter `mxcons`.

Message 42: error - transfer buffer too small in merge1

The buffer used to transfer data between nodes in the MERGE1 subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 45: error - too many atoms in CONFIG file

DL_POLY Classic limits the number of atoms in the system to be simulated and checks for the violation of this condition when it reads the CONFIG file. Termination will result if the condition is violated.
Action:
Standard user response. Fix the parameter `mxatms`. Consider the possibility that the wrong CONFIG file is being used (e.g similar system, but larger size.)

Message 46: error - ewlbuf array too small in ewald1

The `ewlbuf` array used to store structure factor data in subroutine EWALD1 has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxebuf`.

Message 47: error - transfer buffer too small in merge

The buffer used to transfer data between nodes in the MERGE subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 48: error - transfer buffer too small in fortab

The buffer used to transfer data between nodes in the FORTAB subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 49: error - frozen core-shell unit specified

The DL_POLY Classic option to freeze the location of an atom (i.e. hold it permanently in one position) is not permitted for core-shell units. This includes freezing the core or the shell independently.

Action:
Remove the frozen atom option from the FIELD file. Consider using a non-polarisable atom instead.

Message 50: error - too many bond angles specified

DL_POLY Classic limits the number of valence angle potentials that can be specified in the FIELD file and checks for the violation of this. Termination will result if the condition is violated. Do not confuse this error with that described by message 51 (below).

Action:
Standard user response. Fix the parameter `mxtang`.

Message 51: error - too many bond angles in system

DL_POLY Classic limits the number of valence angle potentials in the system to be simulated (actually, the number to be processed by each node) and checks for the violation of this. Termination will result if the condition is violated. Do not confuse this error with that described by message 50 (above).

Action:
Standard user response. Fix the parameter `mxangl`. Consider the possibility that the wrong CONFIG file is being used (e.g similar system, but larger size.)

Message 52: error - end of FIELD file encountered

This message results when DL_POLY Classic reaches the end of the FIELD file, without having read all the data it expects. Probable causes: missing data or incorrect specification of integers on the various directives.

Action:
Check FIELD file for missing or incorrect data and correct.

Message 53: error - end of CONTROL file encountered

This message results when DL_POLY Classic reaches the end of the CONTROL file, without having read all the data it expects. Probable cause: missing **finish** directive.

Action:
Check CONTROL file and correct.

Message 54: error - problem reading CONFIG file

This message results when DL_POLY Classic encounters a problem reading the CONFIG file. Possible cause: corrupt data.

Action:
Check CONFIG file and correct.

Message 55: error - end of CONFIG file encountered

This error arises when DL_POLY Classic attempts to read more data from the CONFIG file than is actually present. The probable cause is an incorrect or absent CONFIG file, but it may be due to the FIELD file being incompatible in some way with the CONFIG file.

Action:
Check contents of CONFIG file. If you are convinced it is correct, check the FIELD file for inconsistencies.

Message 57: error - too many core-shell units specified

DL_POLY Classic has a restriction of the number of types of core-shell unit in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 59 (below).

Action:
Standard user response. Fix the parameter `mxtshl`.

Message 59: error - too many core-shell units in system

DL_POLY Classic limits the number of core-shell units in the simulated system. Termination results if too many are encountered. Do not confuse this error with that described by message 57 (above).

Action:
Standard user response. Fix the parameter `mxshl`.

Message 60: error - too many dihedral angles specified

DL_POLY Classic will accept only a limited number of dihedral angles in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 61 (below).

Action:
Standard user response. Fix the parameter `mxtdih`.

Message 61: error - too many dihedral angles in system

The number of dihedral angles in the whole simulated system is limited by DL_POLY Classic. Termination results if too many are encountered. Do not confuse this error with that described by message 60 (above).

Action:
Standard user response. Fix the parameter `mxdihd`.

Message 62: error - too many tethered atoms specified

DL_POLY Classic will accept only a limited number of tethered atoms in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 63 (below).

Action:
Standard user response. Fix the parameter `mxteth`.

Message 63: error - too many tethered atoms in system

The number of tethered atoms in the simulated system is limited by DL_POLY Classic. Termination results if too many are encountered. Do not confuse this error with that described by message 62 (above).

Action:
Standard user response. Fix the parameter `msteth`.

Message 65: error - too many excluded pairs specified

This error can arise when DL_POLY Classic is identifying the atom pairs that cannot have a pair potential between them, by virtue of being chemically bonded for example (see subroutine EXCLUDE). Some of the working arrays used in this operation may be exceeded, resulting in termination of the program.

Action:
Standard user response. Fix the parameter `mxexcl`.

Message 66: error - incorrect boundary condition for HK ewald

The Hautman-Klein Ewald method can only be used with XY planar periodic boundary conditions (i.e. `imcon` = 6).

Action:
Either the periodic boundary condition, or the choice of calculation of the electrostatic forces must be changed.

Message 67: error - incorrect boundary condition in thbfrc

Three body forces in DL_POLY Classic are only permissible with cubic, orthorhombic and parallelepiped periodic boundaries. Use of other boundary conditions results in this error.

Action:
If nonperiodic boundaries are required, the only option is to use a very large simulation cell, with the required system at the centre surrounded by a vacuum. This is not very efficient however and use of a realistic periodic system is the best option.

Message 69: error - too many link cells required in thbfrc

The calculation of three body forces in DL_POLY Classic is handled by the link cell algorithm. This error arises if the required number of link cells exceeds the permitted array dimension in the code.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 70: error - constraint bond quench failure

When a simulation with bond constraints is started, DL_POLY Classic attempts to extract the kinetic energy of the constrained atom-atom bonds arising from the assignment of initial random

velocities. If this procedure fails, the program will terminate. The likely cause is a badly generated initial configuration.

Action:
Some help may be gained from increasing the cycle limit, by following the standard user response to increase the control parameter `mxshak`. You may also consider reducing the tolerance of the SHAKE iteration, the directive **shake** in the CONTROL file. However it is probably better to take a good look at the starting conditions!

Message 71: error - too many metal potentials specified

The number of metal potentials that can be specfied in the FIELD file is limited. This error results if too many are used.

Action:
Standard user response. Fix the parameter `mxvdw`. Note that this parameter must be *double* the number of required metal potentials. Recompile the program.

Message 72: error - different metal potential types specified

DL_POLY Classic does not permit the user to mix different types of metal potential in the same simulation. There are no known rules for making alloys in this way.
Action:
Change the FIELD (and TABEAM) file as required so that only one type of metal potential is used.

Message 73: error - too many inversion potentials specified

The number of inversion potentials specified in the FIELD file exceeds the permitted maximum.

Action:
Standard user response. Fix the parameter `mxtinv`.

Message 75: error - too many atoms in specified system

DL_POLY Classic places a limit on the number of atoms that can be simulated. Termination results if too many are specified.

Action:
Standard user response. Fix the parameter `mxatms`.

Message 77: error - too many inversion potentials in system

The simulation contains too many inversion potentials overall, causing termination of run.

Action:
Standard user response. Fix the parameter `mxinv`.

Message 79: error - incorrect boundary condition in fbpfrc

The 4-body force routine assumes a cubic or parallelepiped periodic boundary condition is in operation. The job will terminate if this is not adhered to.

Action:
You must reconfigure your simulation to an appropriate boundary condition.

Message 80: error - too many pair potentials specified

DL_POLY Classic places a limit on the number of pair potentials that can be specified in the FIELD file. Exceeding this number results in termination of the program execution.

Action:
Standard user response. Fix the parameters `mxsvdw`. and `mxvdw`.

Message 81: error - unidentified atom in pair potential list

DL_POLY Classic checks all the pair potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 82: error - calculated pair potential index too large

In checking the pair potentials specified in the FIELD file DL_POLY Classic calculates a unique integer index that henceforth identifies the potential within the program. If this index becomes too large, termination of the program results.

Action:
Standard user response. Fix the parameters `mxsvdw` and `mxvdw`.

Message 83: error - too many three body potentials specified

DL_POLY Classic has a limit on the number of three body potentials that can be defined in the FIELD file. This error results if too many are included.

Action:
Standard user response. Fix the parameter `mxtbp`.

Message 84: error - unidentified atom in 3-body potential list

DL_POLY Classic checks all the 3-body potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 85: error - required velocities not in CONFIG file

If the user attempts to start up a DL_POLY Classic simulation with the **restart** or **restart scale** directives (see description of CONTROL file,) the program will expect the CONFIG file to contain atomic velocities as well as positions. Termination results if these are not present.

Action:
Either replace the CONFIG file with one containing the velocities, or if not available, remove the **restart** directive altogether and let DL_POLY Classic create the velocities for itself.

Message 86: error - calculated 3-body potential index too large

DL_POLY Classic has a permitted maximum for the calculated index for any three body potential in the system (i.e. as defined in the FIELD file). If there are m distinct types of atom in the system, the index can possibly range from 1 to $(m^2 * (m-1))/2$. If the internally calculated index exceeds this number, this error report results.

Action:
Standard user response. Fix the parameter `mxtbp`.

Message 87: error - too many link cells required in fbpfrc

The FBPFRC subroutine uses link cells to compute the four body forces. This message indicates that the link cell arrays have insufficient size to work properly.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 88: error - too many tersoff potentials specified

Too many Tersoff potentials have been defined in the FIELD file. Certain arrays must be increased in size to accommodate the data.

Action:
Standard user response. Fix the parameter `mxter`.

Message 89: error - too many four body potentials specified

Too many four body potential have been defined in the FIELD file. Certain arrays must be increased in size to accommodate the data.

Action:
Standard user response. Fix the parameter `mxfbp`.

Message 90: error - system total electric charge nonzero

In DL_POLY Classic a check on the total system charge will result in an error if the net charge of the system is nonzero. (Note: In DL_POLY Classic this message has been disabled. The program merely prints a warning stating that the system is not electrically neutral but it does not terminate the program - watch out for this.)

Action:
Check the specified atomic charges and their populations. Make sure they add up to zero. If the system is required to have a net zero charge, you can enable the call to this error message in subroutine SYSDEF.

Message 91: error - unidentified atom in 4-body potential list

The specification of a four-body potential in the FIELD file has referenced an atom type that is unknown.

Action:
Locate the erroneous atom type in the four body potential definition in the FIELD file and correct. Make sure this atom type is specified by an `atoms` directive earlier in the file.

Message 92: error - unidentified atom in tersoff potential list

The specification of a Tersoff potential in the FIELD file has referenced an atom type that is unknown.

Action:
Locate the erroneous atom type in the Tersoff potential definition in the FIELD file and correct. Make sure this atom type is specified by an `atoms` directive earlier in the file.

Message 93: error - cannot use shell model with rigid molecules

The dynamical shell model implemented in DL_POLY Classic is not designed to work with rigid molecules. This error results if these two options are simultaneously selected.

Action:
In some circumstances you may consider overriding this error message and continuing with your simulation. For example if your simulation does not require the polarisability to be a feature of the rigid species, but is confined to free atoms or flexible molecules in the same system. The appropriate error trap is found in subroutine SYSDEF.

Message 95: error - potential cutoff exceeds half cell width

In order for the minimum image convention to work correctly within DL_POLY Classic, it is necessary to ensure that the cutoff applied to the pair potentials does not exceed half the perpendicular width of the simulation cell. (The perpendicular width is the shortest distance between opposing cell faces.) Termination results if this is detected. In NVE simulations this can only happen at the start of a simulation, but in NPT, it may occur at any time.

Action:
Supply a cutoff that is less than half the cell width. If running constant pressure calculations, use a cutoff that will accommodate the fluctuations in the simulation cell. Study the fluctuations in the OUTPUT file to help you with this.

Message 97: error - cannot use shell model with neutral groups

The dynamical shell model was not designed to work with neutral groups. This error results if an attempt is made to combine both.

Action:
There is no general remedy for this error if you wish to combine both these capabilities. However if your simulation does not require the polarisability to be a feature of rigid species (comprising the charged groups), but is confined to free atoms or flexible molecules in the same system, you may consider overriding this error message and continuing with your simulation. The appropriate error trap is found in subroutine SYSDEF.

Message 99: error - cannot use shell model with constraints

The dynamical shell model was not designed to work in conjunction with constraint bonds. This error results if both are used in the same simulation.

Action: There is no general remedy if you wish to combine both these capabilities. However if your simulation does not require the polarisability to be a feature of the constrained species, but is confined to free atoms or flexible molecules, you may consider overriding this error message and continuing with your simulation. The appropriate error trap is in subroutine SYSDEF.

Message 100: error - forces working arrays too small

There are a number of arrays in DL_POLY Classic that function as workspace for the forces calculations. Their dimension is equal to the number of atoms in the simulation cell divided by the number of nodes. If these arrays are likely to be exceeded, DL_POLY Classic will terminate execution.

Action:
Standard user response. Fix the parameter `msatms`.

Message 101: error - calculated 4-body potential index too large

DL_POLY Classic has a permitted maximum for the calculated index for any four body potential in the system (i.e. as defined in the FIELD file). If there are m distinct types of atom in the system, the index can possibly range from 1 to $(m^2 * (m+1) * (m+2))/6$. If the internally calculated index exceeds this number, this error report results.

Action:
Standard user response. Fix the parameter `mxfbp`.

Message 102: error - parameter mxproc exceeded in shake arrays

The RD-SHAKE algorithm distributes data over all nodes of a parallel computer. Certain arrays in RD-SHAKE have a minimum dimension equal to the maximum number of nodes DL_POLY Classic is likely to encounter. If the actual number of nodes exceeds this, the program terminates.

Action:
Standard user response. Fix the parameter `mxproc`.

Message 103: error - parameter mxlshp exceeded in shake arrays

The RD-SHAKE algorithm requires that information about 'shared' atoms be passed between nodes. If there are too many atoms, the arrays holding the information will be exceeded and

DL_POLY Classic will terminate execution.

Action:
Standard user response. Fix the parameter `mxlshp`.

Message 105: error - shake algorithm failed to converge

The RD-SHAKE algorithm for bond constraints is iterative. If the maximum number of permitted iterations is exceeded, the program terminates. Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow the standard user response to increase the control parameter `mxshak`. But the trouble is much more likely to be cured by careful consideration of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 106: error - neighbour list array too small in parlink

Construction of the Verlet neighbour list in subroutine `parlink` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 107: error - neighbour list array too small in parlinkneu

Construction of the Verlet neighbour list in subroutine `parlinkneu` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 108: error - neighbour list array too small in parneulst

Construction of the Verlet neighbour list in subroutine `parneulst` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 109: error - neighbour list array too small in parlst_nsq

Construction of the Verlet neighbour list in subroutine `parlst_nsq` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 110: error - neighbour list array too small in parlst

Construction of the Verlet neighbour list in subroutine `parlst` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 112: error - vertest array too small

This error results when the dimension of the DL_POLY Classic VERTEST arrays, which are used in checking if the Verlet list needs updating, have been exceeded.

Action:
Standard user response. Fix the parameter `mslst`.

Message 120: error - invalid determinant in matrix inversion

DL_POLY Classic occasionally needs to calculate matrix inverses (usually the inverse of the matrix of cell vectors, which is of size 3×3). For safety's sake a check on the determinant is made, to prevent inadvertent use of a singular matrix.

Action:
Locate the incorrect matrix and fix it - e.g. are cell vectors correct?

Message 130: error - incorrect octahedral boundary condition

When calculating minimum images DL_POLY Classic checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the truncated octahedral minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing cubic cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 135: error - incorrect hexagonal prism boundary condition

When calculating minimum images DL_POLY Classic checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the hexagonal prism minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing orthorhombic cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 140: error - incorrect dodecahedral boundary condition

When calculating minimum images DL_POLY Classic checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the rhombic dodecahedral minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing tetragonal simulation cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 141: error - duplicate metal potential specified

The user has specified a particular metal potential more than once in the FIELD file.

Action:
Locate the metal potential specification in the FIELD file and remove or correct the potential concerned.

Message 142: error - interpolation outside range of metal potential attempted

The program has found that an interatomic distance in a simulated metallic system is such that it requires a potential value outside range for which the potential is defined.

Action:
The probable cause of this is that the density of the system is unrealistic or the potential is being used in unsuitable circumstances. The attempted simulation should be examined, and if considered reasonable a new potential must be found.

Message 145: error - no van der waals potentials defined

This error arises when there are no VDW potentials specified in the FIELD file but the user has not specified **no vdw** in the CONTROL file. In other words DL_POLY Classic expects the FIELD file to contain VDW potential specifications.

Action:
Edit the FIELD file to insert the required potentials or specify **no vdw** in the CONTROL file.

Message 150: error - unknown van der waals potential selected

DL_POLY Classic checks when constructing the interpolation tables for the short ranged potentials that the potential function requested is one which is of a form known to the program. If the requested potential form is unknown, termination of the program results. The most probable cause of this is the incorrect choice of the potential keyword in the FIELD file or one in the wrong columns (input is formatted).

Action:
Read the DL_POLY Classic documentation and find the potential keyword for the potential desired.

Insert the correct index in the FIELD file definition and ensure it occurs in the correct columns (17-20). If the correct form is not available, look at the subroutine FORGEN (or its variant) and define the potential for yourself. It is easily done.

Message 151: error - unknown metal potential selected

The metal potentials available in DL_POLY Classic are confined to density dependent forms of the Sutton-Chen type. This error results if the user attempts to specify another.

Action:
Re-specify the potential as Sutton-Chen type if possible. Check the potential keyword appears in columns 17-20 of the FIELD file.

Message 153: error - metals not permitted with multiple timestep

The multiple timestep algorithm cannot be used in conjunction with metal potentials in DL_POLY Classic.

Action:
The simulation must be run without the multiple timestep option.

Message 160: error - unaccounted for atoms in exclude list

This error message means that DL_POLY Classic has been unable to find all the atoms described in the exclusion list within the simulation cell. This should never occur, if it does it means a serious bookkeeping error has occured. The probable cause is corruption of the code somehow.

Action:
If you feel you can tackle it - good luck! Otherwise we recommend you get in touch with the program authors. Keep all relevant data files to help them find the problem.

Message 170: error - too many variables for statistic array

This error means the statistics arrays appearing in subroutine STATIC are too small. This can happen if the number of unique atom types is too large.

Action:
Standard user response. Fix the parameter `mxnstk`. `mxnstk` should be at least (45+*number of unique atom types*).

Message 180: error - Ewald sum requested in non-periodic system

DL_POLY Classic can use either the Ewald method or direct summation to calculate the electrostatic potentials and forces in periodic (or pseudo-periodic) systems. For non-periodic systems only direct summation is possible. If the Ewald summation is requested (with the **ewald sum** or **ewald precision** directives in the CONTROL file) without periodic boundary conditions, termination of the program results.

Action:
Select periodic boundaries by setting the variable `imcon`>0 in the CONFIG file (if possible) or use a different method to evaluate electrostatic interactions e.g. by usinf the **coul** directive in the CONTROL file.

Message 185: error - too many reciprocal space vectors

DL_POLY Classic places hard limit on the number of k vectors to be used in the Ewald sum and terminates if more than this is requested.

Action:
Either consider using fewer k vectors in the Ewald sum (and a larger cutoff in real space) or follow standard user response to reset the parameters `kmaxb, kmaxc`.

Message 186: error - transfer buffer array too small in sysgen

In the subroutine SYSGEN.F DL_POLY Classic requires dimension of the array `buffer` (defined by the parameter `mxbuff`) to be no less than the parameter `mxatms` or the product of parameters `mxnstk*mxstak`. If this is not the case it will be unable to restart the program correctly to continue a run. (Applies to parallel implementations only.)

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 190: error - buffer array too small in splice

DL_POLY Classic uses a workspace array named `buffer` in several routines. Its declared size is a compromise of several rôles and may sometimes be too small (though in the supplied program, this should happen only very rarely). The point of failure is in the SPLICE routine, which is part of the RD-SHAKE algorithm.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 200: error - rdf buffer array too small in revive

This error indicates that the buffer array used to globally sum the rdf arrays in subroutine REVIVE is too small.

Action:
Standard user response. Fix the parameter `mxbuff`. Alternatively `mxrdf` can be set smaller.

Message 220: error - too many neutral groups in system

DL_POLY Classic has a fixed limit on the number of charged groups in a simulation. This error results if the number is exceeded.

Action:
Standard user response. Fix the parameter `mxneut`.

Message 225: error - multiple selection of optimisation options

The user has specified more than one optimisation directive in the CONTROL file//
Action:
Remove redundant optimisation directive(s) from CONTROL file.

Message 230: error - neutral groups improperly arranged

In the DL_POLY Classic FIELD file the charged groups must be defined in consecutive order. This error results if this convention is not adhered to.

Action:
The arrangement of the data in the FIELD file must be sorted. All atoms in the same group must be arranged consecutively. Note that reordering the file in this way implies a rearrangement of the CONFIG file also.

Message 250: error - Ewald sum requested with neutral groups

DL_POLY Classic will not permit the use of neutral groups with the Ewald sum. This error results if the two are used together.

Action:
Either remove the **neut** directive from the FIELD file or use a different method to evaluate the electrostatic interactions.

Message 260: error - parameter mxexcl exceeded in excludeneu routine

An error has been detected in the construction of the excluded atoms list for neutral groups. This occurs when the parameter `mxexcl` is exceeded in the EXCLUDENEU routine.

Action:
Standard user response. Fix parameter `mxexcl`.

Message 300: error - incorrect boundary condition in parlink

The use of link cells in DL_POLY Classic implies the use of appropriate boundary conditions. This error results if the user specifies octahedral, dodecahedral or slab boundary conditions.

Action:
The simulation must be run with cubic, orthorhombic or parallelepiped boundary conditions.

Message 301: error - too many rigid body types

The maximum number of rigid body types permitted by DL_POLY Classic has been exceeded.

Action:
Standard user response. Fix the parameter `mxungp`.

Message 302: error - too many sites in rigid body

This error arises when DL_POLY Classic finds that the number of sites in a rigid body exceeds the dimensions of the approriate storage arrays.

Action:
Standard user response. Fix the parameter `mxngp`.

Message 303: error - too many rigid bodies specified

The maximum number of rigid bodies in a simulation has been reached. Do not confuse this with message 304 below.

Action:
Standard user response. Fix the parameter `mxgrp`.

Message 304: error - too many rigid body sites in system

This error occurs when the total number of sites within all rigid bodies exceeds the permitted maximum. Do not confuse this with message 303 above.

Action:
Standard user response. Fix the parameter `mxgatms`.

Message 305: error - box size too small for link cells

The link cells algorithm in DL_POLY Classic cannot work with less than 27 link cells. Depending on the cell size and the chosen cut-off, DL_POLY Classic may decide that this minimum cannot be achieved and terminate.

Action:
If a smaller cut-off is acceptable use it. Otherwise do not use link cells. Consider running a larger system, where link cells will work.

Message 306: error - failed to find principal axis system

This error indicates that the routine QUATBOOK has failed to find the principal axis for a rigid unit.

Action:
This is an unlikely error. The code should correctly handle linear, planar and 3-dimensional rigid units. Check the definition of the rigid unit in the CONFIG file, if sensible report the error to the authors.

Message 310: error - quaternion setup failed

This error indicates that the routine QUATBOOK has failed to reproduce all the atomic positions in rigid units from the centre of mass and quaternion vectors it has calculated.

Action:
Check the contents of the CONFIG file. DL_POLY Classic builds its local body description of a rigid unit type from the *first* occurrence of such a unit in the CONFIG file. The error most likely occurs because subsequent occurrences were not sufficiently similar to this reference structure. If the problem persists increase the value of the variable `tol` in QUATBOOK and recompile. If problems still persist double the value of `dettest` in QUATBOOK and recompile. If you still encounter problems contact the authors.

Message 320: error - site in multiple rigid bodies

DL_POLY Classic has detected that a site is shared by two or more rigid bodies. There is no integration algorithm available in this version of the package to deal with this type of model.

Action:
The only course is to redefine the molecular model (e.g. introducing flexible bonds and angles in suitable places) to allow DL_POLY Classic to proceed.

Message 321: error - quaternion integrator failed

The quaternion algorithm has failed to converge. If the maximum number of permitted iterations is exceeded, the program terminates. Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. Try reducing the timestep or running a zero kelvin structure optimization for a hundred timesteps or so. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow the standard user response to increase the parameter `mxquat`. But the trouble is much more likely to be cured by careful consideration of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 330: error - mxewld parameter incorrect

DL_POLY Classic has two strategies for parallelization of the reciprocal space part of the Ewald sum. If EWALD1 is used the parameter `mxewld` should equal the parameter `msatms`. If EWALD1A is used this parameter should equal `mxatms`.

Action:
Standard user response. Set the parameter `mxewld` to the value appropriate for the version of EWALD1 you are using. Recompile the program.

Message 331: error - mxhke parameter incorrect

The parameter `mxhke`, which defines the dimension of some arrays used in the Hautman-Klein Ewald method, should equal the parameter `msatms`.

Action:
Standard user response. Set the parameter `mxhke` to the value regquired. Recompile the program.

Message 332: error - mxhko parameter too small

The parameter `mxhko` defines the maximum order for the Taylor expansion implicit in the Hautman-Klein Ewald method. DL_POLY Classic has a maximum of `mxhko` = 3, but it can be set to less in some implementations. If this error arises when the user requestes an order in excess of this parameter.

Action:
Standard user response. Set the parameter `mxhko` to a higher value (if it is <3) and recompile the

program. Alternatively request a lower order in the CONTROL file through the `nhko` variable (see 4.1.1).

Message 340: error - invalid integration option requested

DL_POLY Classic has detected an incompatibility in the simulation instructions, namely that the requested integration algorithm is not compatible with the physical model. It *may* be possible to override this error trap, but it is up to the user to establish if this is sensible. *Action:*
This is a non recoverable error, unless the user chooses to override the restriction.

Message 350: error - too few degrees of freedom

This error can arise if a small system is being simulated and the number of constraints applied is too large.

Action:
Simulate a larger system or reduce the number of constraints.

Message 360: error - frozen atom found in rigid body

DL_POLY Classic does not permit a site in a rigid body to be frozen i.e. fixed in one location in space.

Action:
Remove the 'freeze' condition from the site concerned. Consider using a very high site mass to achieve a similar effect.

Message 380: error - simulation temperature not specified

DL_POLY Classic has failed to find a **temp** directive in the CONTROL file.

Action:
Place a **temp** directive in the CONTROL file, with the required temperature specified.

Message 381: error - simulation timestep not specified

DL_POLY Classic has failed to find a **timestep** directive in the CONTROL file.

Action:
Place a **timestep** directive in the CONTROL file, with the required timestep specified.

Message 382: error - simulation cutoff not specified

DL_POLY Classic has failed to find a **cutoff** directive in the CONTROL file.

Action:
Place a **cutoff** directive in the CONTROL file, with the required forces cutoff specified.

Message 383: error - simulation forces option not specified

DL_POLY Classic has failed to find any directive specifying the electrostatic interactions options in the CONTROL file.

Action:
Ensure the CONTROL file contains at least one directive specifying the electrostatic potentials (e.g. **ewald**, **coul**, **no electrostatics** etc.)

Message 384: error - verlet strip width not specified

DL_POLY Classic has failed to find the **delr** directive in the CONTROL file.

Action:
Insert a **delr** directive in the CONTROL file, specifying the width of the verlet strip augmenting the forces cutoff.

Message 385: error - primary cutoff not specified

DL_POLY Classic has failed to find the **prim** directive in the CONTROL file. Necessary only if multiple timestep option required.

Action:
Insert a **prim** directive in the CONTROL file, specifying the primary cutoff radius in the multiple timestep algorithm.

Message 386: error - primary cutoff larger than rcut

The primary cutoff specified by the **prim** directive in the CONTROL file exceeds the value specified for the forces cutoff (directive **cut**). Applies only if the multiple timestep option is required.

Action:
Locate the **prim** directive in the CONTROL file, and alter the chosen cutoff. Alternatively, increase the real space cutoff specified with the **cut** directive. Take care to avoid error number 398.

Message 387: error - system pressure not specified

The target system pressure has not been specified in the CONTROL file. Applies to NPT simulations only.

Action:
Insert a **press** directive in the CONTROL file specifying the required system pressure.

Message 388: error - npt incompatible with multiple timestep

The use of NPT (constant pressure) and temperature is not compatible with the multiple timestep option.

Action:
Simulation must be run at fixed volume in this case. But note it may be possible to use NPT without the multiple timestep, in ourder to estimate the required system volume, then switch back to multiple timestep and NVT dynamics at the required volume.

Message 390: error - npt ensemble requested in non-periodic system

A non-periodic system has no defined volume, hence the NPT algorithm cannot be applied.

Action:
Either simulate the system with a periodic boundary, or use another ensemble.

Message 391: error - incorrect number of pimd beads in config file

The CONFIG file must specify the position of all the beads in a PIMD simulation, not just the positions of the parent atoms, otherwise this error results.

Action:
The CONFIG file must be reconstructed to provide the required data.

Message 392: error - too many link cells requested

The number of link cells required for a given simulation exceeds the number allowed for by the DL_POLY Classic arrays.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 394: error - minimum image arrays exceeded

The work arrays used in IMAGES have been exceeded.
Action: Standard user response. Fix the parameter `mxxdf`.

Message 396: error - interpolation array exceeded

DL_POLY Classic has sought to read past the end of an interpolation array. This should never happen!

Action:
Contact the authors.

Message 398: error - cutoff too small for rprim and delr

This error can arise when the multiple timestep option is used. It is essential that the primary cutoff (`rprim`) is less than the real space cutoff (`rcut`) by at least the Verlet shell width `delr` (preferably much larger!). DL_POLY Classic terminates the run if this condition is not satisfied.

Action:
Adjust `rcut`, `rprim` and `delr` to satisfy the DL_POLY Classic requirement. These are defined with the directives **cut**, **prim** and **delr** respectively.

Message 400: error - rvdw greater than cutoff

DL_POLY Classic requires the real space cutoff (`rcut`) to be larger than, or equal to, the van der Waals cutoff (`rvdw`) and terminates the run if this condition is not satisfied.

Action:
Adjust `rvdw` and `rcut` to satisfy the DL_POLY Classic requirement.

Message 402: error - van der waals cutoff unset

The user has not set a cutoff (`rvdw`) for the van der Waals potentials. The simulation cannot proceed without this being specified.

Action:
Supply a cutoff value for the van der Waals terms in the CONTROL file using the directive **rvdw**, and resubmit job.

Message 410: error - cell not consistent with image convention

The simulation cell vectors appearing in the CONFIG file are not consistent with the specified image convention.

Action:
Locate the variable `imcon` in the CONFIG file and correct to suit the cell vectors.

Message 412: error - mxxdf parameter too small for shake routine

In DL_POLY Classic the parameter `mxxdf` must be greater than or equal to the parameter `mxcons`. If it is not, this error is a possible result.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 414: error - conflicting ensemble options in CONTROL file

DL_POLY Classic has found more than one **ensemble** directive in the CONTROL file.

Action:
Locate extra **ensemble** directives in CONTROL file and remove.

Message 416: error - conflicting force options in CONTROL file

DL_POLY Classic has found incompatible directives in the CONTROL file specifying the electrostatic interactions options.

Action:
Locate the conflicting directives in the CONTROL file and correct.

Message 418: error - bond vector work arrays too small in bndfrc

The work arrays in BNDFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 419: error - bond vector work arrays too small in angfrc

The work arrays in ANGFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 420: error - bond vector work arrays too small in tethfrc

The work arrays in TETHFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 421: error - bond vector work arrays too small in dihfrc

The work arrays in DIHFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 422: error - all-pairs must use multiple timestep

In DL_POLY Classic the 'all pairs' option must be used in conjunction with the multiple timestep.

Action:
Activate the multiple timestep option in the CONTROL file and resubmit.

Message 423: error - bond vector work arrays too small in shlfrc

The dimensions of the interatomic distance vectors have been exceeded in subroutine SHLFRC.

Action:
Standard user response. Fix the parameter `msbad`. Set equal to the value of the parameter `mxshl`.

Message 424: error - electrostatics incorrect for all-pairs

When using the all pairs option in conjunction with electrostatic forces, the electrostatics must be handled with either the standard Coulomb sum, or with the distance dependent dielectric.

Action:
Rerun the simulation with the appropriate electrostatic option.

Message 425: error - transfer buffer array too small in shlmerge

The buffer used to transfer data between nodes in the subroutine SHLMERGE has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 426: error - neutral groups not permitted with all-pairs

DL_POLY Classic will not permit simulations using both the neutral group and all pairs options together.

Action:
Switch off one of the conflicting options and rerun.

Message 427: error - bond vector work arrays too small in invfrc

The work arrays in subroutine INVFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 430: error - integration routine not available

A request for a nonexistent ensemble has been made or a request with conflicting options that DL_POLY Classic cannot deal with (e.g. a Evans thermostat with rigid body equations of motion).

Action:
Examine the CONTROL and FIELD files and remove inappropriate specifications.

Message 432: error - intlist failed to assign constraints

If the required simulation has constraint bonds DL_POLY Classic attempts to apportion the molecules to processors so that, if possible, there are no shared atoms between processors. If this is not possible, one or more molecules may be split between processors. This message indicates that the code has failed to carry out either of these successfully.

Action:
The error may arise from a compiler error. Try recompiling INTLIST without the optimization flag turned on. If the problem persists it should be reported to the authors, (after checking the input data for inconsistencies).

Message 433: error - specify rcut before the Ewald sum precision

When specifying the desired precision for the Ewald sum in the CONTROL file, it is first necessary to specify the real space cutoff `rcut`.

Action:
Place the **cut** directive *before* the **ewald precision** directive in the CONTROL file and rerun.

Message 434: error - illegal entry into STRESS related routine

The calculation of the stress tensor in DL_POLY Classic requires additional code that must be included at compile time through the use of the STRESS keyword. If this is not done, and DL_POLY Classic is later required to calculate the stress tensor, this error will result.

Action:
The program must be recompiled with the STRESS keyword activated. This will ensure all the relevant code is in place. See section 3.2.1.

Message 435: error - specify rcut before the coulomb precision

When specifying the desired precision for the coulomb sum in the CONTROL file, it is first necessary to specify the real space cutoff `rcut`.

Action:
Place the **cut** directive *before* the **coulomb precision** directive in the CONTROL file and rerun.

Message 436: error - unrecognised ensemble

An unknown ensemble option has been specified in the CONTROL file.

Action:
Locate **ensemble** directive in the CONTROL file and amend appropriately.

Message 438: error - PMF constraints failed to converge

The constraints in the potential of mean force algorithm have not converged in the permitted number of cycles. (The SHAKE algorithm for PMF constraints is iterative.) Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow standard user response to increase the parameter `mxshak`. But the trouble is much more likely to be cured by careful consideration of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 440: error - undefined angular potential

A form of angular potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is possible. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and ANGFRC will be required.

Message 442: error - undefined three body potential

A form of three body potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and THBFRC will be required.

Message 443: error - undefined four body potential

DL_POLY Classic has been requested to process a four-body potential it does not recognise.

Action:
Check the FIELD file and make sure the keyword is correctly defined. Make sure that subroutine

FBPFRC contains the code necessary to deal with the requested potential. Add the code required if necessary, by amending subroutines SYSDEF and FBPFRC.

Message 444: error - undefined bond potential

DL_POLY Classic has been requested to process a bond potential it does not recognise.

Action:
Check the FIELD file and make sure the keyword is correctly defined. Make sure that subroutine BNDFRC contains the code necessary to deal with the requested potential. Add the code required if necessary, by amending subroutines SYSDEF and BNDFRC.

Message 445: error - undefined many body potential

DL_POLY Classic has been requested to process a many body potential it does not recognise.

Action:
Check the FIELD file and make sure the keyword is correctly defined. Make sure the code version you are using contains the code necessary to deal with the requested potential. Add the code required if necessary.

Message 446: error - undefined electrostatic key in dihfrc

The subroutine DIHFRC has detected a request for an unknown kind of electrostatic model.

Action:
The probable source of the error is an improperly described force field. Check the CONTROL file and FIELD files for incompatible requirements.

Message 447: error - 1-4 separation exceeds cutoff range

In the subroutine DIHFRC the distance between the 1-4 atoms in the potential is larger than the cutoff that is applied to the 1-4 potential, meaning the potential will not be computed, though it may be an essential component of the dihedral force and not necessarily a vanishing force.

Action:
The probable source of the error is an improperly described force field. Effectively the 1-4 distance is not being restrained sufficently. Check the 1-4 potential parameters and the valence angles that help define the dihedral geometry. If these are correct, then you may have to comment out this error condition in the DIHFRC.F subroutine, but beware that when the 1-4 atoms are too widely separated, the dihedral angle can become indeterminable.

Message 448: error - undefined dihedral potential

A form of dihedral potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to

DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and DIHFRC (and its variants) will be required.

Message 449: error - undefined inversion potential

A form of inversion potential has been encountered which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and INVFRC will be required.

Message 450: error - undefined tethering potential

A form of tethering potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and TETHFRC will be required.

Message 451: error - three body potential cutoff undefined

The cutoff radius for a three body potential has not been defined in the FIELD file.

Action:
Locate the offending three body force potential in the FIELD file and add the required cutoff. Resubmit the job.

Message 452: error - undefined pair potential

A form of pair potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and FORGEN will be required.

Message 453: error - four body potential cutoff undefined

The cutoff radius for a four-body potential has not been defined in the FIELD file.

Action:
Locate the offending four body force potential in the FIELD file and add the required cutoff. Resubmit the job.

Message 454: error - undefined external field

A form of external field potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and EXTNFLD will be required.

Message 456: error - core and shell in same rigid unit

It is not sensible to fix both the core and the shell of a polarisable atom in the same molecular unit. Consequently DL_POLY Classic will abandon the job if this is found to be the case.

Action:
Locate the offending core-shell unit (there may be more than one in your FIELD file) and release the shell (preferably) from the rigid body specification.

Message 458: error - too many PMF constraints - param. mspmf too small

The number of constraints in the potential of mean force is too large. The dimensions of the appropriate arrays in DL_POLY Classic must be increased.

Action:
Standard user response. Fix the parameter `mspmf`.

Message 460: error - too many PMF sites - parameter mxspmf too small

The number of sites defined in the potential of mean force is too large. The dimensions of the appropriate arrays in DL_POLY Classic must be increased.

Action:
Standard user response. Fix the parameter `mxspmf`.

Message 461: error - undefined metal potential

The user has requested a metal potential DL_POLY Classic does not recognise.

Action:
Locate the metal potential specification in the FIELD file and replace with a recognised potential.

Message 462: error - PMF UNIT record expected

A **pmf unit** directive was expected as the next record in the FIELD file but was not found.

Action:
Locate the **pmf** directive in the FIELD file and examine the following entries. Insert the missing **pmf unit** directive and resubmit.

Message 463: error - unidentified atom in metal potential list

DL_POLY Classic checks all the metal potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 464: error - thermostat time constant must be > 0.d0

A zero or negative value for the thermostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 465: error - calculated pair potential index too large

A zero or negative value for the thermostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 466: error - barostat time constant must be > 0.d0

A zero or negative value for the barostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 468: error - r0 too large for snm potential with current cutoff

The specified location (r0) of the potential minimum for a shifted n-m potential exceeds the specified potential cutoff. A potential with the desired minimum cannot be created.

Action:
To obtain a potential with the desired minimum it is necessary to increase the van der Waals cutoff. Locate the **rvdw** directive in the CONTROL file and reset to a magnitude greater than r0. Alternatively adjust the value of r0 in the FIELD file. Check that the FIELD file is correctly formatted.

Message 470: error - n<m in definition of n-m potential

The specification of a n-m potential in the FIELD file implies that the exponent m is larger than exponent n. (Not all versions of DL_POLY Classic are affected by this.)

Action:
Locate the n-m potential in the FIELD file and reverse the order of the exponents. Resubmit the job.

Message 474: error - mxxdf too small in parlst subroutine

The parameter `mxxdf` defining working arrays in subroutine PARLST of DL_POLY Classic has been found to be too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 475: error - mxxdf too small in parlst_nsq subroutine

The parameter `mxxdf` defining working arrays in subroutine PARLST_NSQ DL_POLY Classic has been found to be too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 476: error - mxxdf too small in parneulst subroutine

The parameter `mxxdf` defining working arrays in subroutine PARNEULST is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 477: error - mxxdf too small in prneulst subroutine

The parameter `mxxdf` defining working arrays in subroutine PRNEULST is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 478: error - mxxdf too small in forcesneu subroutine

The parameter `mxxdf` defining working arrays in subroutine FORCESNEU is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 479: error - mxxdf too small in multipleneu subroutine

The parameter `mxxdf` defining working arrays in subroutine MULTIPLENEU is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 484: error - only one potential of mean force permitted

It is not permitted to define more than one potential of mean force in the FIELD file.

Action:
Check that the FIELD file contains only one PMF specification. If more than one is needed, DL_POLY Classic cannot handle it.

Message 486: error - HK real space screening function cutoff violation

DL_POLY Classic has detected an unacceptable degree of inaccuracy in the screening function near the radius of cutoff *in real space*, which implies the Hautman-Klein Ewald method will not be sufficiently accurate.

Action:
The user should respecify the HK control parameters given in the CONTROL file. Either the convergence parameter should be increased or the sum expanded to incorporate more images of the central cell. (Warning: increasing the convergence parameter may cause failure in the reciprocal space domain.) (See 4.1.1).

Message 487: error - HK recip space screening function cutoff violation

DL_POLY Classic has detected an unacceptable degree of inaccuracy in the screening function near the radius of cutoff *in reciprocal space*, which implies the Hautman-Klein Ewald method will not be sufficiently accurate.

Action:
The user should respecify the HK control parameters given in the CONTROL file. Either the convergence parameter should be reduced or more k-vectors used. (Warning: reducing the convergence parameter may cause failure in the real space domain.) (See 4.1.1).

Message 488: error - HK lattice control parameter set too large

The Hautman-Klein Ewald method in DL_POLY Classic permits the user to perform a real space sum over nearest-neighbour and next-nearest-neighbour cells (i.e. up to `nlatt=2`). If the user specifies a larger sum than this, this error will result.

Action:
The user should respecify the HK control parameters given in the CONTROL file and set `nlatt` to a maximum of 2. (See 4.1.1).

Message 490: error - PMF parameter mxpmf too small in passpmf

The bookkeeping arrays have been exceeded in PASSPMF

Action:
Standard user response. Fix the parameter `mxpmf`. Set equal to `mxatms`.

Message 492: error - parameter mxcons < number of PMF constraints

The parameter `mxcons` is too small for the number of PMF constraints in the system.

Action:
Standard user response. Fix the value of `mxcons`.

Message 494: error in csend: pvmfinitsend

The PVM routine PVMFINITSEND has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 496: error in csend: pvmfpack

The PVM routine PVMFPACK has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 498: error in csend: pvmfsend

The PVM routine PVMFSEND has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 500: error in crecv: pvmfrecv

The PVM routine PVMFRECV has returned an error. It is invoked by the routine CRECV.

Action:
Check your system implementation of PVM.

Message 502: error in crecv: pvmfunpack

The PVM routine PVMFUNPACK has returned an error. It is invoked by the routine CRECV.

Action:
Check your system implementation of PVM.

Message 504: error - cutoff too large for TABLE file

The requested cutoff exceeds the information in the TABLE file.

Action:
Reduce the value of the vdw cutoff (`rvdw`) in the CONTROL file or reconstruct the TABLE file.

Message 506: error - work arrays too small for quaternion integration

The working arrays associated with quaternions are too small for the size of system being simulated. They must be redimensioned.

Action:
Standard user response. Fix the parameter `msgrp`.

Message 508: error - rigid bodies not permitted with RESPA algorithm

The RESPA algorithm implemented in DL_POLY Classic is for atomic systems only. Rigid bodies or constraints cannot be treated.

Action:
There is no cure for this. The code simply does not have this capability. Consider writing it for yourself!

Message 510: error - structure optimiser not permitted with RESPA

The RESPA algorithm in DL_POLY Classic has not been implemented to work with the structure optimizer. You have asked for a forbidden operation.

Action:
There is no fix for this. In any case it does not make sense to use the RESPA algorithm for this purpose.

Message 513: error - SPME not available for given boundary conditions

The SPME algorithm in DL_POLY Classic does not work for aperiodic (IMCON=0) or slab (IMCON=6) boundary conditions.
Action:
If the system must have aperiodic or slab boundaries, nothing can be done. In the latter case however, it may be acceptable to represent the same system with slabs replicated in the z direction, thus permitting a periodic simulation.

Message 514: error - SPME routines have not been compiled in

The inclusion of the SPME algorithm in DL_POLY Classic is optional at the compile stage. If the executable does not contain the SPME routines, but the method is requested by the user, this error results.

Action:
DL_POLY Classic must be recompiled with the SPME flags set. Beware that your system has the necessary fast Fourier transform routines to permit this.

Message 516: error - repeat of impact option specified

More than one impact option has been specified in the CONTROL file. Only one is allowed.

Action:
Remove the offending impact directive from the CONTROL file and rerun.

Message 601: error - Ewald SPME incompatible with solvation

The options in DL_POLY Classic that use the energy decomposition/solvation facility do not permit the use of the SPME option. It is possible however to use the standard Ewald method.

Action:
Change the SPME directive in the CONTROL file to ewald and rerun.

Message 602: error - Ewald HK incompatible with solvation

The options in DL_POLY Classic that use the energy decomposition/solvation facility do not permit the use of the Hautman-Klein Ewald option. It is possible however to use the standard Ewald method.

Action:
Change the HKE directive in the CONTROL file to ewald. Make sure the system model includes a large vacuum gap between material slabs to offset the effects of the periodic boundary.

Message 1000: error - failed allocation of configuration arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1010: error - failed allocation of angle arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1011: error - failed allocation of dihedral arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1012: error - failed allocation of exclude arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1013: error - failed allocation of rigid body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1014: error - failed allocation of vdw arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1015: error - failed allocation of lr correction arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1020: error - failed allocation of angle work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1030: error - failed allocation of bond arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1040: error - failed allocation of bond work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1050: error - failed allocation of dihedral arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1060: error - failed allocation of dihedral work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1070: error - failed allocation of constraint arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1090: error - failed allocation of site arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1100: error - failed allocation of core_shell arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1115: error - failed allocation of hyperdynamics work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1010: error - failed allocation of angle arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1120: error - failed allocation of inversion arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1130: error - failed allocation of inversion work arrays'

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1140: error - failed allocation of four-body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1150: error - failed allocation of four-body work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1170: error - failed allocation of three-body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1180: error - failed allocation of three-body work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1200: error - failed allocation of external field arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1210: error - failed allocation of pmf arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1220: error - failed allocation of pmf_lf or pmf_vv work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1230: error - failed allocation of pmf_shake work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1240: error - failed allocation of ewald arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1250: error - failed allocation of excluded atom arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1260: error - failed allocation of tethering arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1270: error - failed allocation of tethering work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1280: error - failed allocation of metal arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1290: error - failed allocation of work arrays in nvt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1300: error - failed allocation of dens0 array in npt_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1310: error - failed allocation of work arrays in npt_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1320: error - failed allocation of dens0 array in npt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1330: error - failed allocation of work arrays in npt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1340: error - failed allocation of dens0 array in nst_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1350: error - failed allocation of work arrays in nst_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1360: error - failed allocation of dens0 array in nst_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1370: error - failed allocation of work arrays in nst_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1380: error - failed allocation of work arrays in nve_1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1390: error - failed allocation of work arrays in nvt_e1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1400: error - failed allocation of work arrays in nvt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1410: error - failed allocation of work arrays in nvt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1420: error - failed allocation of work arrays in npt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1430: error - failed allocation of density array in npt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1440: error - failed allocation of work arrays in npt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1450: error - failed allocation of density array in npt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1460: error - failed allocation of work arrays in nst_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1470: error - failed allocation of density array in nst_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1480: error - failed allocation of work arrays in nst_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1490: error - failed allocation of density array in nst_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1500: error - failed allocation of work arrays in nveq_1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1510: error - failed allocation of work arrays in nvtq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1520: error - failed allocation of work arrays in nvtq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1530: error - failed allocation of work arrays in nptq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1540: error - failed allocation of density array in nptq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1550: error - failed allocation of work arrays in nptq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1560: error - failed allocation of density array in nptq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1570: error - failed allocation of work arrays in nstq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1580: error - failed allocation of density array in nstq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1590: error - failed allocation of work arrays in nstq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1600: error - failed allocation of density array in nstq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1610: error - failed allocation of work arrays in qshake.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1615: error - failed allocation of work arrays in qrattle_q.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1620: error - failed allocation of work arrays in nveq_2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1625: error - failed allocation of work arrays in qrattle_v.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1630: error - failed allocation of work arrays in nvtq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1640: error - failed allocation of work arrays in nvtq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1650: error - failed allocation of work arrays in nptq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1660: error - failed allocation of density array in nptq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1670: error - failed allocation of work arrays in nptq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1680: error - failed allocation of density array in nptq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1690: error - failed allocation of work arrays in nstq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1700: error - failed allocation of density array in nstq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1710: error - failed allocation of work arrays in nstq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1720: error - failed allocation of density array in nstq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1730: error - failed allocation of HK Ewald arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1740: error - failed allocation of property arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1750: error - failed allocation of spme arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1760: error - failed allocation of ewald_spme.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1770: error - failed allocation of quench.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1780: error - failed allocation of quatqnch.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1790: error - failed allocation of quatbook.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1800: error - failed allocation of intlist.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1810: error - failed allocation of forces.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1820: error - failed allocation of forcesneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1830: error - failed allocation of neutlst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1840: error - failed allocation of multiple.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1850: error - failed allocation of multipleneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1860: error - failed allocation of multiple_nsq.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1870: error - failed allocation of parlst_nsq.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1880: error - failed allocation of parlst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1890: error - failed allocation of parlink.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1900: error - failed allocation of parlinkneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1910: error - failed allocation of parneulst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1920: error - failed allocation of zero_kelvin.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1925: error - failed allocation of strucopt.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1930: error - failed allocation of vertest.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1940: error - failed allocation of pair arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1945: error - failed allocation of tersoff arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1950: error - shell relaxation cycle limit exceeded

There has been a convergence failure during the execution of relaxed shell polarisation model. Probable cause: the system is unstable e.g. in an abnormally high energy configuration.
Action:
Increasing the maximum number of cycles permitted in the shell relaxation set by variable `mxpass` in the dlpoly.f root program may help, but it is unlikely. A better option is to relax the structure somehow first e.g. using the **zero** option in the CONTROL file.

Message 1951: error - no shell dynamics algorithm specified

The user has failed to specify which of the available shell dynamics algorithm is to be used in the simulation. Options include adiabtic shells and relaxed shells.
Action:
Locate the definition of the core-shell units in the FIELD file and check that all necessary integer keys have been supplied. Consult the user manual if in doubt.

Message 1953: error - tersoff radius of cutoff not defined

The Tersoff potential requires the user to specify a short ranged cutoff as part of the potential description. This is distinct from the normal cutoff used by the Van der Waals interactions.
Action:
Check the Tersoff potential description in the FIELD file. Make sure it is fully complete.

Message 1955: error - failed allocation of tersoff work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1960: error - conflicting shell option in FIELD file

The relaxed shell and adiabatic shell polarisation options in DL_POLY Classic are mutually exclusive. The user has request both options in the same simulation.
Action:
Locate the occurrences of the **shell** directives in the FIELD file and ensure they specify the same shell model.

Message 1970: error - failed allocation of shell_relax work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1972: error - unknown tersoff potential defined in FIELD file

DL_POLY Classic has failed to recognise the Tersoff potentials specified by the user in the FIELD file.

Action:
Locate the Tersoff potential specification in the FIELD fiel and ensure it is correctly defined.

Message 1974: error - incorrect period boundary in tersoff.f

The implementation of the Tersoff potential in DL_POLY Classic is based on the link cell algorithm, which is suitable for rectangular or triclinic MD cells only. It is not suitable for any other shape of MD cell.
Action:
The user must reconstruct the system according to one of the permitted periodic boundaries.

Message 1976: error - too many link cells required in tersoff.f

The number of link cells required by the Tersoff routines exceeds the amount allowed for by DL_POLY Classic. This can happen if the system is simulated under NPT or NST conditions and the system volume increases dramatically.
Action:
The problem may cure itself on restart, provided the restart configuration has already expande significantly. Otherwise the user must locate and adjust the `mxcell` according to the standard response procedure.

Message 1978: error - undefined potential in tersoff.f

A form of Tersoff potential has been requested which DL_POLY Classic does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY Classic if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and TERSOFF will be required.

Message 1980: error - failed allocation of nvevv_1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1990: error - failed allocation of nvtvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2000: error - failed allocation of nvtvv_e1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2010: error - failed allocation of nvtvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2020: error - failed allocation of nptvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2030: error - failed allocation of nptvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2040: error - failed allocation of nptvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2050: error - failed allocation of nptvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2060: error - failed allocation of nstvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2070: error - failed allocation of nstvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2080: error - failed allocation of nstvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2090: error - failed allocation of nstvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2100: error - failed allocation of nveqvv_1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2110: error - failed allocation of nveqvv_2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2120: error - failed allocation of nvtqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2130: error - failed allocation of nvtqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2140: error - failed allocation of nvtqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2150: error - failed allocation of nvtqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2160: error - failed allocation of nptqvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2170: error - failed allocation of nptqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2180: error - failed allocation of nptqvv_b2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2190: error - failed allocation of nptqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2200: error - failed allocation of nptqvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2210: error - failed allocation of nptqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2220: error - failed allocation of nptqvv_h2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2230: error - failed allocation of nptqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2240: error - failed allocation of nstqvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2250: error - failed allocation of nstqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2260: error - failed allocation of nstqvv_b2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2270: error - failed allocation of nstqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2280: error - failed allocation of nstqvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2290: error - failed allocation of nstqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2300: error - failed allocation of nstqvv_h2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2310: error - failed allocation of nstqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2320: error - NEB convergence failure

The nudged elastic band calculation in the temperature accelerated dynamics or bias potential dynamics has failed to converge.

Action:
The best approach is to halt the TAD or BPD simulation and focus on the NEB calculation in isolation. First try to reproduce the error by a straightforward NEB calculation using the same start and end points for the chain. Adjusting the convergence criteria may offer a way forward. Try minimising the start and end points independently to a higher precision. It is possible that the start and end points are too far apart, so that one or more intermedate states have been missed. This leads to multiple maxima on the reaction path, which may be the problem. In which case examine the operational choices made in running the TAD or BPD simulation and see if changing them will reduce the danger of this happening.

Message 2330: error - too many basin files found - increase mxbsn

A TAD or BPD run has generated more than 100 basin files, which is the internal operational limit.

Action:
Reset the *mxbsn* parameter, which is defined at the top of the hyper_dynamics_module.f file, to a larger number and recompile.

Message 2340: error - TAD diffs arrays exceeded - increase mxdiffs

A TAD or BPD run has generated more than 300 recorded differences between the reference structure and all subsequent new basins found. Effectively this means it has recorded more than 300

atomic jumps, which is the internal operational limit.

Action:
Reset the *mxdiffs* parameter, which is defined at the top of the hyper_dynamics_module.f file, to a larger number and recompile.

Message 2350: error - kinks found in NEB chain during optimisation

During a TAD or BPD run the nudged elastic band calculation is unable to converge because kinking of the chain has occurred.

Action:
Tricky. This implies there is something extreme about the system potential energy surface, such as it having an excessive number of undulations, or perhaps the simulation has start and end states are too far apart. This may be fixed by trying different operational parameters, such as using a different number of beads in the NEB chain, or perhaps the simulation is being run at too high a temperature. Some experimentation is required, but it may be possible that the system just isn't suitable for investigation by TAD or BPD.

Message 2355: error - cannot run both TAD and BPD together

The TAD and BPD options are not meant to run concurrently. Choose one or the other!

Action:
Remove either the TAD or BPD option from the CONTROL file.

Message 2500: error in number of collective variables - ncolvar too small?

The number of order parameters in a metadynamics simulation has not been properly specified.
Action:
Check input data in CONTROL file and correct accordingly.

Message 2501: Wang-Landau style recursion not yet implemented for ncolvar > 1

The Wang-Landau recursion option in metadynamics is currently limted to one order parameter only.
Action:
Select another Gaussian convergence option in the CONTROL file.

Message 2502: Unrecognised Gaussian height scheme

An invalid option has been selected for the metadynamics Gaussian convergence scheme, which is rstricted to values 0,1 and 2,
Action:
Reset the **hkey** option to an acceptable value in the CONTROL file.

Message 2503: Error maxhis exceeded in metadynamics

The internal storage of Gaussian data in metadynamics has been exceeded.
Action:
This can be recovered if a greater number of processing nodes is used at restart, but ideally, a less ambitious Gaussian deposition rate should be considered.

Message 2504: Error allocating comms buffer in compute_bias_potential

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2505: Error allocating driven array

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2508: Comms error in metadynamics setup

This is probably a programming error and should not occur.
Action:
Identify and fix the bug if you can. Otherwise locate the authors and ask for a fix.

Message 2509: Cannot bias local and global PE in same run

The metadynamics option does not allow the use of both global and local potential energy order parameters at the same time.
Action:
Decide which of these options you really need and reset the directives in the CONTROL file.

Message 2510: Error allocating local force arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2511: Error allocating collective variables arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2512: Error allocating Wang-Landau bins

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2515: Error allocating Steinhardt parameter arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2516: Could not open STEINHARDT

The STEINHARDT data (input) file cannot be opened.
Action:
The file is probably not available, or is unreadable. Restore the file as required and rerun.

Message 2517: Error allocating q4site

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2518: Error allocating q6site

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2519: Error deallocating buff

Unlikely array deallocation error, which should not occur under normal use.
Action:
Possible system error. Raise issue with system manager.

Message 2521: Error reading line _ of STEINHARDT

The nominated line of the STEINHARDT file cannot be read.
Action:
Probably missing or corrupted data line in file. Locate and correct.

Message 2522: Error allocating Steinhardt parameter arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2523: Could not open ZETA

The ZETA data (input) file cannot be opened.
Action:
The file is probably not available, or is unreadable. Restore the file as required and rerun.

Message 2524: Error allocating zetasite

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2525: Error allocating full neighbour list

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2527: Number of collective variables incorrect for specified order parameters

The internal check of the requested number of order parameters in a metadynamics simulation has found an inconsistency.
Action:
Check the total number of collective variables (ncolvar) matches total number specified by nq4, nq6, ntet and potential energy parameters.

Message 2529: Error reading line _ of ZETA

There has been an error reading the nominated line of the ZETA file.
Action:
Probably a missing or corrupted data line. Locate and fix.

Message 2531: Comms error on reading METADYNAMICS

This is probably a programming error and should not occur.
Action:
Identify and fix the bug if you can. Otherwise locate the authors and ask for a fix.

Message 2532: Error in fc function - out of range

The switching function has been incorrectly defined in a hyperdynamics simulation.
Action:
Check the value reported and make the necessary correction in the STEINHARDT or ZETA file concerned.

Message 2533: Error allocating solvation arrays for metadynamics

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2534: Error allocating comms buffer arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2535: Solvation list overrun

The arrays tabulating the coordination list for either Steinhardt or tetrahedral order parameters have been exceeded.
Action:
Locate the specification of the variable *maxneigh* in the metafreeze_module.f file (there are 3 occurrences) and reset to a larger number.

Message 2536: Error deallocating solvation arrays for metadynamics

Unlikely array deallocation error, which should not occur under normal use.
Action:
Possible system error. Raise issue with system manager.

Message 2537: Error deallocating comms buffer arrays

Unlikely array deallocation error, which should not occur under normal use.
Action:
Possible system error. Raise issue with system manager.

Message 2538: Error allocating solvation arrays for metadynamics

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2540: Error allocating force prefactor arrays

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2541: Memory allocation error in compute_tet_nlist

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2542: Error in metafreeze_module.f90 mxninc too small

The internal estimate of the array allocation variable *mxninc* is too small for the purpose.
Action:
Locate where variable is defined in metafreeze_module.f and reset to a larger number.

Message 2543: nnn too small in compute_tet_nlist

The internal estimate of the array allocation variable *nnn* is too small for the purpose.
Action:
Locate where variable is defined in metafreeze_module.f and reset to a larger number.

Message 2544: mxflist too small in metafreeze_module

The internal estimate of the array allocation variable *mxflist* is too small for the purpose.
Action:
Locate where variable is defined in metafreeze_module.f and reset to a larger number.

Message 2545: Memory deallocation error in compute_tet_nlist

Unlikely array deallocation error, which should not occur under normal use.
Action:
Probable system error. Raise issue with system manager.

Message 2546: Memory allocation error in compute_tet_nlist

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Message 2547: Memory deallocation error in compute_tet_nlist

Unlikely array deallocation error, which should not occur under normal use.
Action:
Probable system error. Raise issue with system manager.

Message 2548: Memory allocation error in compute_tet_nlist

Unlikely array allocation error, which should not occur under normal use.
Action:
The user is probably making excessive demands of memory. Reconsider the problem size in relation to compute resource.

Appendix D

Subroutine Locations

The Locations of Subroutines and Functions

The following table lists the subroutines and functions in DL_POLY Classic and which source files they can be found in.

Routine	Kind	Location
abort_config_read	subroutine	define_system_module.f
abort_control_read	subroutine	define_system_module.f
abort_eamtable_read	subroutine	metal_module.f
abort_field_read	subroutine	define_system_module.f
abort_table_read	subroutine	vdw_module.f
abortscan	subroutine	setup_module.f
alloc_ang_arrays	subroutine	angles_module.f
alloc_bnd_arrays	subroutine	bonds_module.f
alloc_config_arrays	subroutine	config_module.f
alloc_csh_arrays	subroutine	core_shell_module.f
alloc_dih_arrays	subroutine	dihedral_module.f
alloc_ewald_arrays	subroutine	ewald_module.f
alloc_exc_arrays	subroutine	exclude_module.f
alloc_exi_arrays	subroutine	solvation_module.f
alloc_fbp_arrays	subroutine	four_body_module.f
alloc_fld_arrays	subroutine	external_field_module.f
alloc_free_arrays	subroutine	solvation_module.f
alloc_hke_arrays	subroutine	hkewald_module.f
alloc_hyper_arrays	subroutine	hyper_dynamics_module.f
alloc_inv_arrays	subroutine	inversion_module.f
alloc_met_arrays	subroutine	metal_module.f
alloc_pair_arrays	subroutine	pair_module.f
alloc_pmf_arrays	subroutine	pmf_module.f
alloc_prp_arrays	subroutine	property_module.f
alloc_rgbdy_arrays	subroutine	rigid_body_module.f
alloc_shake_arrays	subroutine	shake_module.f
alloc_site_arrays	subroutine	site_module.f
alloc_sol_arrays	subroutine	solvation_module.f
alloc_spme_arrays	subroutine	spme_module.f
alloc_tbp_arrays	subroutine	three_body_module.f

```
alloc_ter_arrays           subroutine  tersoff_module.f
alloc_tet_arrays           subroutine  tether_module.f
alloc_vdw_arrays           subroutine  vdw_module.f
angfrc                     subroutine  angles_module.f
bndfrc                     subroutine  bonds_module.f
bodystress                 subroutine  rigid_body_module.f
bomb                       subroutine  utility_module.f
bpd_forces                 subroutine  hyper_dynamics_module.f
bpd_option                 subroutine  define_system_module.f
bspcoe                     subroutine  spme_module.f
bspgen                     subroutine  spme_module.f
cell_propagate             subroutine  ensemble_tools_module.f
cell_update                subroutine  ensemble_tools_module.f
cerfr                      function    hkewald_module.f
cfgscan                    subroutine  setup_module.f
check_basins               subroutine  hyper_dynamics_module.f
check_for_transition       subroutine  hyper_dynamics_module.f
check_shells               subroutine  core_shell_module.f
check_syschg               subroutine  site_module.f
compute_bias_potential     subroutine  metafreeze_module.f
comput_steinhardt          subroutine  metafreeze_module.f
compute_steinhardt_forces  subroutine  metafreeze_module.f
compute_tet_nlist          subroutine  metafreeze_module.f
compute_tetrahedral        subroutine  metafreeze_module.f
compute_tetrahedral_forces subroutine  metafreeze_module.f
config_write               subroutine  utility_module.f
conscan                    subroutine  setup_module.f
copy_force                 subroutine  solvation_module.f
copystring                 subroutine  parse_module.f
corshl                     subroutine  core_shell_module.f
coul0                      subroutine  coulomb_module.f
coul0neu                   subroutine  neu_coul_module.f
coul1                      subroutine  coulomb_module.f
coul2                      subroutine  coulomb_module.f
coul2neu                   subroutine  neu_coul_module.f
coul3                      subroutine  coulomb_module.f
coul3neu                   subroutine  neu_coul_module.f
coul4                      subroutine  coulomb_module.f
coul_nsq                   subroutine  coulomb_module.f
cpy_rtc                    subroutine  utility_module.f
crecv                      subroutine  basic_comms.f
crecv                      subroutine  serial.f
csend                      subroutine  basic_comms.f
csend                      subroutine  serial.f
dblstr                     function    parse_module.f
dcell                      subroutine  setup_module.f
define_angles              subroutine  angles_module.f
define_atoms               subroutine  site_module.f
define_bonds               subroutine  bonds_module.f
```

define_constraints	subroutine	shake_module.f
define_core_shell	subroutine	core_shell_module.f
define_dihedrals	subroutine	dihedral_module.f
define_external_field	subroutine	external_field_module.f
define_four_body	subroutine	four_body_module.f
define_inversions	subroutine	inversion_module.f
define_metadynamics	subroutine	metafreeze_module.f
define_metals	subroutine	metal_module.f
define_minimum_state	subroutine	hyper_dynamics_module.f
define_pmf	subroutine	pmf_module.f
define_rigid_body	subroutine	rigid_body_module.f
define_tersoff	subroutine	tersoff_module.f
define_tethers	subroutine	tether_module.f
define_three_body	subroutine	three_body_module.f
define_units	subroutine	define_system_module.f
define_van_der_waals	subroutine	vdw_module.f
deposit_gaussian	subroutine	metafreeze_module.f
dfc	function	metafreeze_module.f
diffsn0	subroutine	property_module.f
diffsn1	subroutine	property_module.f
dihfrc	subroutine	dihedral_module.f
dlpfft3	subroutine	spme_module.f
duni	function	utility_module.f
eamden	subroutine	metal_module.f
ele_prd	subroutine	utility_module.f
energy_unit	function	define_system_module.f
ensemble_selection	subroutine	define_system_module.f
erfcgen	subroutine	ewald_module.f
error	subroutine	error_module.f
ewald1	subroutine	ewald_module.f
ewald2	subroutine	ewald_module.f
ewald3	subroutine	ewald_module.f
ewald4	subroutine	ewald_module.f
ewald_selection	subroutine	define_system_module.f
ewald_spme	subroutine	spme_module.f
excitation_option	subroutine	define_system_module.f
exclude	subroutine	exclude_module.f
exclude_atom	subroutine	exclude_module.f
exclude_link	subroutine	exclude_module.f
excludeneu	subroutine	exclude_module.f
exitcomms	subroutine	basic_comms.f
exitcomms	subroutine	serial.f
extnfld	subroutine	external_field_module.f
fbpfrc	subroutine	four_body_module.f
fc	function	metafreeze_module.f
fcap	subroutine	utility_module.f
findstring	function	parse_module.f
fldscan	subroutine	setup_module.f
force_manager	subroutine	forces_module.f

forces	subroutine	forces_module.f
forces_neu	subroutine	forces_module.f
forgen	subroutine	vdw_module.f
fortab	subroutine	vdw_module.f
free_energy_option	subroutine	define_system_module.f
free_energy_write	subroutine	solvation_module.f
free_kinetic	subroutine	solvation_module.f
freegen	subroutine	solvation_module.f
freeze	subroutine	utility_module.f
fsden	subroutine	metal_module.f
gauss	subroutine	utility_module.f
gdsum	subroutine	basic_comms.f
gdsum	subroutine	serial.f
get_prntime	subroutine	utility_module.f
get_simtime	subroutine	utility_module.f
getcom	subroutine	ensemble_tools_module.f
getcom_mol	subroutine	utility_module.f
getkin	function	ensemble_tools_module.f
getkinf	function	ensemble_tools_module.f
getking	subroutine	ensemble_tools_module.f
getkinr	function	ensemble_tools_module.f
getkins	subroutine	ensemble_tools_module.f
getkint	function	ensemble_tools_module.f
getmass	function	ensemble_tools_module.f
getrec	subroutine	parse_module.f
getrotmat	subroutine	utility_module.f
getvom	subroutine	ensemble_tools_module.f
getword	subroutine	parse_module.f
gimax	subroutine	basic_comms.f
gimax	subroutine	serial.f
gisum	subroutine	basic_comms.f
gisum	subroutine	serial.f
global_sum_forces	subroutine	utility_module.f
gstate	subroutine	basic_comms.f
gstate	subroutine	serial.f
gsync	subroutine	basic_comms.f
gsync	subroutine	serial.f
hkewald1	subroutine	hkewald_module.f
hkewald2	subroutine	hkewald_module.f
hkewald3	subroutine	hkewald_module.f
hkewald4	subroutine	hkewald_module.f
hkgen	subroutine	hkewald_module.f
hyper_close	subroutine	hyper_dynamics_module.f
hyper_driver	subroutine	hyper_dynamics_module.f
hyper_open	subroutine	hyper_dynamics_module.f
hyper_start	subroutine	hyper_dynamics_module.f
images	subroutine	utility_module.f
impact	subroutine	temp_scalers_module.f
initcomms	subroutine	basic_comms.f

initcomms	subroutine	serial.f
intlist	subroutine	define_system_module.f
intstr	function	parse_module.f
intstr3	function	utility_module.f
invert	subroutine	utility_module.f
invfrc	subroutine	inversion_module.f
jacobi	subroutine	utility_module.f
kinstr	subroutine	ensemble_tools_module.f
kinstress	subroutine	ensemble_tools_module.f
kinstressf	subroutine	ensemble_tools_module.f
kinstressg	subroutine	ensemble_tools_module.f
lf_integrate	subroutine	integrator_module.f
loc2	function	utility_module.f
loc3	function	utility_module.f
loc4	function	utility_module.f
lowcase	subroutine	parse_module.f
lrcmetal	subroutine	metal_module.f
lrcorrect	subroutine	vdw_module.f
lrcorrect_fre	subroutine	solvation_module.f
lrcorrect_sol	subroutine	solvation_module.f
machine	subroutine	basic_comms.f
machine	subroutine	serial.f
mat_mul	subroutine	utility_module.f
merge	subroutine	merge_tools.f
merge	subroutine	serial.f
merge1	subroutine	merge_tools.f
merge1	subroutine	serial.f
merge4	subroutine	merge_tools.f
merge4	subroutine	serial.f
metafreeze_driver	subroutine	metafreeze_module.f
metal_deriv	subroutine	metal_module.f
metdens	subroutine	metal_module.f
metfrc	subroutine	metal_module.f
metgen	subroutine	metal_module.f
mettab	subroutine	metal_module.f
mfrz_error	subroutine	metafreeze_module.f
minimiser	subroutine	driver_module.f
mkwd8	function	parse_module.f
molecular_dynamics	subroutine	driver_module.f
multiple	subroutine	forces_module.f
multiple_neu	subroutine	forces_module.f
multiple_nsq	subroutine	forces_module.f
mynode	function	basic_comms.f
mynode	function	serial.f
neb_driver	subroutine	hyper_dynamics_module.f
neb_option	subroutine	define_system_module.f
neb_spring_forces	subroutine	hyper_dynamics_module.f
neb_system_forces	subroutine	hyper_dynamics_module.f
neutbook	subroutine	define_system_module.f

neutlst	subroutine	forces_module.f
nlist_driver	subroutine	nlist_builders_module.f
nodedim	function	basic_comms.f
nodedim	function	serial.f
nosquish	subroutine	vv_rotation1_module.f
npt_b1	subroutine	lf_motion_module.f
npt_h1	subroutine	lf_motion_module.f
nptq_b1	subroutine	lf_rotation1_module.f
nptq_b2	subroutine	lf_rotation2_module.f
nptq_h1	subroutine	lf_rotation1_module.f
nptq_h2	subroutine	lf_rotation2_module.f
nptqscl_p	subroutine	ensemble_tools_module.f
nptqscl_t	subroutine	ensemble_tools_module.f
nptqvv_b1	subroutine	vv_rotation1_module.f
nptqvv_b2	subroutine	vv_rotation2_module.f
nptqvv_h1	subroutine	vv_rotation1_module.f
nptqvv_h2	subroutine	vv_rotation2_module.f
nptscale_p	subroutine	ensemble_tools_module.f
nptscale_t	subroutine	ensemble_tools_module.f
nptvv_b1	subroutine	vv_motion_module.f
nptvv_h1	subroutine	vv_motion_module.f
nst_b1	subroutine	lf_motion_module.f
nst_h1	subroutine	lf_motion_module.f
nstq_b1	subroutine	lf_rotation1_module.f
nstq_b2	subroutine	lf_rotation2_module.f
nstq_h1	subroutine	lf_rotation1_module.f
nstq_h2	subroutine	lf_rotation2_module.f
nstqmtk_p	subroutine	ensemble_tools_module.f
nstqscl_p	subroutine	ensemble_tools_module.f
nstqscl_p2	subroutine	ensemble_tools_module.f
nstqscl_t	subroutine	ensemble_tools_module.f
nstqscl_t2	subroutine	ensemble_tools_module.f
nstqvv_b1	subroutine	vv_rotation1_module.f
nstqvv_b2	subroutine	vv_rotation2_module.f
nstqvv_h1	subroutine	vv_rotation1_module.f
nstqvv_h2	subroutine	vv_rotation2_module.f
nstscale_p	subroutine	ensemble_tools_module.f
nstscale_t	subroutine	ensemble_tools_module.f
nstvv_b1	subroutine	vv_motion_module.f
nstvv_h1	subroutine	vv_motion_module.f
numnodes	function	basic_comms.f
numnodes	function	serial.f
nve_1	subroutine	lf_motion_module.f
nveq_1	subroutine	lf_rotation1_module.f
nveq_2	subroutine	lf_rotation2_module.f
nveqvv_1	subroutine	vv_rotation1_module.f
nveqvv_2	subroutine	vv_rotation2_module.f
nvevv_1	subroutine	vv_motion_module.f
nvt_b1	subroutine	lf_motion_module.f

nvt_e1	subroutine	lf_motion_module.f
nvt_h1	subroutine	lf_motion_module.f
nvtq_b1	subroutine	lf_rotation1_module.f
nvtq_b2	subroutine	lf_rotation2_module.f
nvtq_h1	subroutine	lf_rotation1_module.f
nvtq_h2	subroutine	lf_rotation2_module.f
nvtqscl	subroutine	ensemble_tools_module.f
nvtqvv_b1	subroutine	vv_rotation1_module.f
nvtqvv_b2	subroutine	vv_rotation2_module.f
nvtqvv_h1	subroutine	vv_rotation1_module.f
nvtqvv_h2	subroutine	vv_rotation2_module.f
nvtscale	subroutine	ensemble_tools_module.f
nvtvv_b1	subroutine	vv_motion_module.f
nvtvv_e1	subroutine	vv_motion_module.f
nvtvv_h1	subroutine	vv_motion_module.f
optimisation_selector	subroutine	optimiser_module.f
parlink	subroutine	nlist_builders_module.f
parlinkneu	subroutine	nlist_builders_module.f
parlst	subroutine	nlist_builders_module.f
parlst_nsq	subroutine	nlist_builders_module.f
parneulst	subroutine	nlist_builders_module.f
parset	subroutine	setup_module.f
passcon	subroutine	pass_tools.f
passcon	subroutine	serial.f
passpmf	subroutine	pass_tools.f
passpmf	subroutine	serial.f
passquat	subroutine	pass_tools.f
passquat	subroutine	serial.f
pivot	subroutine	vv_rotation2_module.f
pmf_rattle_r	subroutine	pmf_module.f
pmf_rattle_v	subroutine	pmf_module.f
pmf_shake	subroutine	pmf_module.f
pmf_vectors	subroutine	pmf_module.f
pmflf	subroutine	pmf_module.f
pmflfq_1	subroutine	pmf_module.f
pmfvv	subroutine	pmf_module.f
primlst	subroutine	nlist_builders_module.f
print_optim	subroutine	define_system_module.f
prneulst	subroutine	nlist_builders_module.f
pseudo_shake	subroutine	optimiser_module.f
put_shells_on_cores	subroutine	core_shell_module.f
qrattle_r	subroutine	vv_rotation2_module.f
qrattle_v	subroutine	vv_rotation2_module.f
qshake	subroutine	lf_rotation2_module.f
quatbook	subroutine	define_system_module.f
quatqnch	subroutine	temp_scalers_module.f
quench	subroutine	temp_scalers_module.f
rdf0	subroutine	property_module.f
rdf0neu	subroutine	property_module.f

rdf1	subroutine	property_module.f
rdrattle_r	subroutine	vv_motion_module.f
rdrattle_v	subroutine	vv_motion_module.f
rdshake_1	subroutine	lf_motion_module.f
read_reference_config	subroutine	hyper_dynamics_module.f
regauss	subroutine	temp_scalers_module.f
relax_shells	subroutine	core_shell_module.f
result	subroutine	property_module.f
revive	subroutine	property_module.f
rotate_omega	subroutine	vv_rotation1_module.f
scan_profile	subroutine	hyper_dynamics_module.f
scl_csum	subroutine	utility_module.f
scramble_velocities	subroutine	hyper_dynamics_module.f
sdot0	function	utility_module.f
sdot1	function	utility_module.f
set_block	subroutine	utility_module.f
shell_relaxation	subroutine	driver_module.f
shellsort	subroutine	utility_module.f
shlfrc	subroutine	core_shell_module.f
shlmerge	subroutine	merge_tools.f
shlmerge	subroutine	serial.f
shlqnch	subroutine	temp_scalers_module.f
shmove	subroutine	merge_tools.f
shmove	subroutine	serial.f
simdef	subroutine	define_system_module.f
solva_temp	subroutine	solvation_module.f
solvation_option	subroutine	define_system_module.f
solvation_write	subroutine	solvation_module.f
spl_cexp	subroutine	spme_module.f
splice	subroutine	merge_tools.f
splice	subroutine	serial.f
spme_for	subroutine	spme_module.f
srfrce	subroutine	vdw_module.f
srfrceneu	subroutine	vdw_module.f
static	subroutine	property_module.f
store_config	subroutine	hyper_dynamics_module.f
strip	subroutine	parse_module.f
striptext	subroutine	parse_module.f
strucopt	subroutine	optimiser_module.f
switch	subroutine	solvation_module.f
switch_atm	subroutine	solvation_module.f
switching_option	subroutine	define_system_module.f
sysbook	subroutine	define_system_module.f
sysdef	subroutine	define_system_module.f
sysgen	subroutine	define_system_module.f
sysinit	subroutine	define_system_module.f
systemp	subroutine	define_system_module.f
tad_option	subroutine	define_system_module.f
tergen	subroutine	tersoff_module.f

terint	subroutine	tersoff_module.f
tersoff	subroutine	tersoff_module.f
tersoff3	subroutine	tersoff_module.f
tethfrc	subroutine	tether_module.f
thbfrc	subroutine	three_body_module.f
timchk	subroutine	utility_module.f
torque_split	subroutine	optimiser_module.f
traject	subroutine	utility_module.f
traject_u	subroutine	utility_module.f
transition_properties	subroutine	hyper_dynamics_module.f
transition_time	subroutine	hyper_dynamics_module.f
turn_rigid_body	subroutine	optimiser_module.f
update_ghost	subroutine	solvation_module.f
update_quaternions	subroutine	lf_rotation1_module.f
vertest	subroutine	nlist_builders_module.f
vertest2	subroutine	nlist_builders_module.f
vscaleg	subroutine	temp_scalers_module.f
vv_integrate	subroutine	integrator_module.f
warning	subroutine	error_module.f
write_profile	subroutine	hyper_dynamics_module.f
write_reference_confi	subroutine	hyper_dynamics_module.f
xscale	subroutine	tether_module.f
zden0	subroutine	property_module.f
zden1	subroutine	property_module.f
zero_kelvin	subroutine	optimiser_module.f

Index

algorithm, 5, 54, 96
  Brode-Ahlrichs, 14, 76, 77
  FIQA, 5, 55, 69
  multiple timestep, 74–76, 99, 101, 102, 131
  NOSQUISH, 5, 56, 69, 70
  QSHAKE, 5, 55, 56, 71, 73, 80
  RATTLE, 5, 56, 58, 79
  SHAKE, 5, 55, 75, 79, 80
  velocity Verlet, 5, 54, 56, 59
  Verlet, 14, 15, 30, 43, 54, 57, 60, 62, 64, 79
  Verlet leapfrog, 5, 54, 55, 57
AMBER, 4, 13
angular momentum, 69
angular restraints, 20
angular velocity, 69

barostat, 5, 71, 98
  Berendsen, 66, 73
  Hoover, 63
BASINS directory, 156
bias potential dynamics (BPD), *see* hyperdynamics, BPD
boundary conditions, 4, 43, 243
  cubic, 106
  hexagonal prism, 106
  rhombic dodecahedron, 106
  truncated octahedron, 106

CCP5, 3
CFGBSNnn file, 156
CFGMIN file, 132
CFGTRKnn file, 157
charge groups, 108
CONFIG file, 104
constraints
  bond, 3, 5, 14, 15, 57, 59, 67, 70, 71, 79, 80, 111, 131
  Gaussian, 46, 47, 59
  PMF, 59, 112
CONTROL file, 96
CVS, 6

direct Coulomb sum, 43
  distance dependent dielectric, 45, 46, 52, 97, 103
  Fennel and Gezelter method, 45
  truncated and shifted, 44
  Wolf method, 45
distance restraints, 17

embedded atom potential, *see* potential,embedded atom (EAM)
energy decomposition, 163
ensemble, 5
  Berendsen NσT, 5, 55, 56, 98, 101, 103
  Berendsen NPT, 5, 55, 56, 101, 103
  Berendsen NVT, 5, 55, 56, 97, 98, 101, 103
  canonical, 59
  Evans NVT, 5, 55, 56, 97, 101, 103
  Hoover NσT, 5, 55, 56, 101
  Hoover NPT, 5, 55, 56, 98, 101
  Hoover NVT, 5, 55, 56, 101
  microcanonical, *see* ensemble,NVE
  NVE, 59, 97, 101, 103
equations of motion
  Euler, 69
  rigid body, 69
error messages, 92, 248
EVENTS file, 155
Ewald
  Hautman Klein, 43, 50, 91, 98, 246
  optimisation, 89, 90
  SPME, 6, 43, 48, 78, 89, 99
  summation, 43, 46–48, 74, 75, 77, 89–91, 98, 100, 101

FIELD file, 107
Finnis-Sinclair potential, *see* potential,Finnis-Sinclair
force field, 4, 13–15, 22, 41, 42
  AMBER, 4, 13
  Ceramics, 187
  DL_POLY, 4, 13, 54
  Dreiding, 4, 13, 30, 31, 187
  GROMOS, 4, 13

OPLS, 13, 187
FORGE, 9
FORTRAN 90, 5–7
free energy
thermodynamic integration, 163, 166
FREENG file, 170

Graphical User Interface, 4, 9, 105
GROMOS, 4, 13
Gupta potential, *see* potential,Gupta

Hautman Klein Ewald, *see* Ewald, Hautman Klein
HISTORY file, 127
formatted, 127
unformatted, 128
hyperdynamics
BPD, 138, 140
exploring configuration space, 147
full path kinetics, 143
NEB, 139, 158
reaction path, 139, 157, 158
TAD, 138, 147

Java GUI, 187

long ranged corrections
metal, 38
van der Waals, 30

metadynamics, 176
collective variables, *see* hyperdynamics, order parameters
Gaussian potential, 176
METADYNAMICS file, 182
order parameter scaling, 179
order parameters, 177–179, 183
potential energy, 177
running simulations, 179
STEINHARDT file, 181
Steinhardt parameters, 178
Tetrahedral parameters, 179
theory of, 176
ZETA file, 181
minimisation, 87
conjugate gradients, 5, 54, 88
programmed, 5, 88
zero temperature, 5, 88

nudged elastic band (NEB), *see* hyperdynamics, NEB

OUTPUT file, 129

parallelisation, 5, 75
Ewald summation, 78
intramolecular terms, 76
Replicated Data, 5
Verlet neighbour list, 77
potential
bond, 4, 14–17, 21, 22, 27, 31, 54, 76, 79, 111, 130
calcite, 25, 26
Coulombic, *see* potential,electrostatic
dihedral, 4, 13, 14, 20–24, 75, 76, 114, 130
electrostatic, 4, 7, 14, 17, 19, 42, 43, 74, 97–99, 103, 130
embedded atom (EAM), 34, 35, 118, 125
Finnis-Sinclair, 34, 35, 118, 119
four-body, 4, 13–15, 28, 34, 78, 117, 118, 130
Gupta, 35, 119
improper dihedral, 4, 23, 75
intramolecular, 28, 34
inversion, 4, 13, 24, 25, 34, 115
metal, 4, 13, 34, 118
nonbonded, 4, 14, 15, 76, 78, 86, 99, 108, 111, 113, 116
Sutton-Chen, 35, 119
tabulated, 124
Tersoff, 13, 14, 31, 33, 120, 121
tethered, 26, 27, 116, 130, 131
three-body, 4, 13–15, 17, 28, 30, 31, 78, 117, 130, 131
valence angle, 4, 13, 14, 17–19, 24, 30, 31, 75, 78, 112, 113, 130
van der Waals, 14, 17, 19, 74, 86, 101, 114
PROFILES directory, 157
PROnn.XY file, 157

quaternions, 5, 55, 69, 99

RDFDAT file, 133
reaction field, 52, 53, 99
REVCON file, 132
REVIVE file, 133
REVOLD file, 122
rigid body, 3, 5, 27, 55–57, 67, 68, 70, 71, 80
rigid bond, *see* constraints,bond

shell model polarisation, 53, 54
dynamical shell model, 53, 54
relaxed shell model, 54
SOLVAT file, 164
solvation energy, 163, *see* energy decomposition

spectroscopic excitation, 163, 171
SPME, *see* Ewald,SPME
STATIS file, 134
stress tensor, 19, 22, 25, 27, 30, 31, 33, 34, 38, 44–46, 48, 53, 54, 57, 58, 63
sub-directory, 234–238
  build, 8
  data, 8
  execute, 8
  java, 8
  source, 8
  utility, 8
Sutton-Chen potential, *see* potential,Sutton-Chen

TABEAM file, 125
TABLE file, 124
temperature accelerated dynamics (TAD), *see* hyperdynamics, TAD
thermostat, 5, 42, 71, 74, 98
  Berendsen, 66, 67, 71, 73
  Nosé-Hoover, 63, 64, 71, 73
TRACKS directory, 157

units
  DL_POLY, 7, 131
  energy, 108
  pressure, 7, 8, 63, 99, 131

Verlet neighbour list, 48, 74, 76–78, 102

WWW, 3, 6, 10

ZDNDAT file, 133